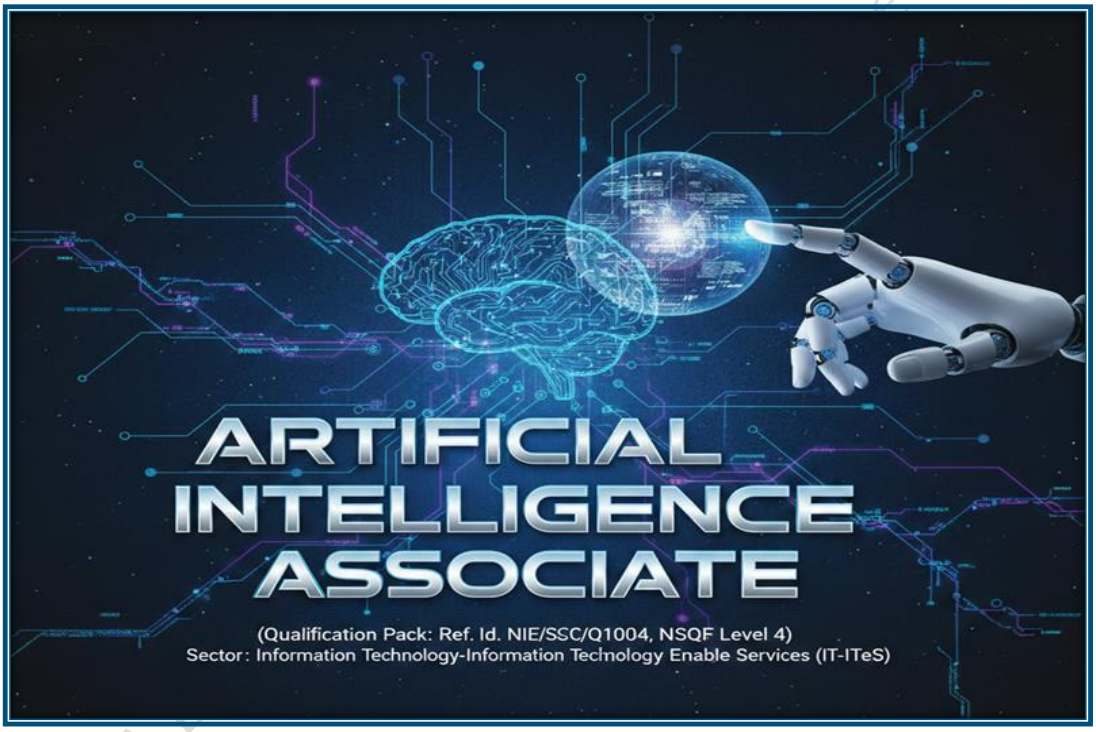
Draft Study Material

ARTIFICIAL INTELLIGENCE ASSOCIATE

(Qualification Pack: Ref. Id. NIE/SSC/Q1004, NSQF Level 4)
Sector: Information Technology-Information Technology Enable Services (IT-ITeS)

(Grade XI)



विद्यया ऽमृतमश्नुते
एन सी ई आर टी
NCERT

PSS CENTRAL INSTITUTE OF VOCATIONAL EDUCATION
(a constituent unit of NCERT, under Ministry of Education, Government of India)
Shyamla Hills, Bhopal- 462 002, M.P., India
https://www.psscive.ac.in

Preface

Vocational Education is a dynamic and evolving field, and ensuring that every student has access to quality learning materials is of paramount importance. The journey of the PSS Central Institute of Vocational Education (PSSCIVE) toward producing comprehensive and inclusive study material is rigorous and time-consuming, requiring thorough research, expert consultation, and publication by the National Council of Educational Research and Training (NCERT). However, the absence of finalized study material should not impede the educational progress of our students. In response to this necessity, we present the draft study material, a provisional yet comprehensive guide, designed to bridge the gap between teaching and learning, until the official version of the study material is made available by the NCERT. The draft study material provides a structured and accessible set of materials for teachers and students to utilize in the interim period. The content is aligned with the prescribed curriculum to ensure that students remain on track with their learning objectives.

The contents of the modules are curated to provide continuity in education and maintain the momentum of teaching-learning in vocational education. It encompasses essential concepts and skills aligned with the curriculum and educational standards. We extend our gratitude to the academicians, vocational educators, subject matter experts, industry experts, academic consultants, and all other people who contributed their expertise and insights to the creation of the draft study material.

Teachers are encouraged to use the draft modules of the study material as a guide and supplement their teaching with additional resources and activities that cater to their students' unique learning styles and needs. Collaboration and feedback are vital; therefore, we welcome suggestions for improvement, especially by the teachers, in improving upon the content of the study material.

Deepak Paliwal
(Joint Director)
PSSCIVE, Bhopal

STUDY MATERIAL DEVELOPMENT COMMITTEE

Members

Akshya Arya, Assistant Professor, Department of Artificial Intelligence & Data Science, Arya College of Engineering, Main Campus, SP40, Kukas, Jaipur (Rajsthan)

Ganesh Kumar Dixit, Professor & Head, Department of Artificial Intelligence & Data Science, Arya College of Engineering, Main Campus, SP40, Kukas, Jaipur (Rajsthan)

Prakash Khanale, Professor and Head, Department of Computer Science, DSM College, Parbhani, Maharashtra

Member Coordinator

Deepak D. Shudhalwar, Professor (CSE), Head, Department of Engineering and Technology, PSSCIVE, NCERT, Bhopal, Madhya Pradesh

CONTENT

Module 1. Introduction to Artificial Intelligence (AI)

Module Overview

Artificial Intelligence (AI) is the technology that enables machines to act and think like humans. It helps computers understand language, recognize images, make predictions, and even solve problems. Today, AI has become part of our daily lives. For example, when you ask Google Assistant or Siri about the weather, AI listens to your voice, understands your question, and replies instantly. When you watch videos on YouTube or Netflix, AI recommends what you might like next, based on your past choices. Apps like Google Maps use AI to suggest the fastest route by analysing live traffic data. In healthcare, AI helps doctors detect diseases early by studying medical images such as X-rays or MRIs.

This Module introduces the concept, history, types, and applications of AI, along with its role in school education and different industries. You will also learn about the responsibilities of AI Associates, the tools and frameworks used to build AI systems, and the future career opportunities in this exciting field.

By the end of the Module, you will:

- Understand what AI is and how it works.
- Explore how AI is applied in real life.
- Recognize the importance of AI in education, healthcare, finance, and smart homes.
- Become aware of the skills needed for future careers in AI.

AI is not just about machines replacing humans—it is about humans and machines working together to solve problems and make life easier.

Fig. 1.1: Artificial Intelligence in our Life

Learning Outcomes

After completing this module, you will be able to:

- Explain the concept of Artificial Intelligence and understand its basic meaning and importance in daily life.

- Identify and differentiate the types of AI such as Narrow AI, General AI, and Super AI.
- Describe the major domains of AI like Machine Learning, Natural Language Processing, and Computer Vision.
- Recognize real-life applications of AI in education, healthcare, banking, and smart devices.
- Understand the role and impact of AI in modern technology and society.

Module Structure

Session 1. Artificial Intelligence

Session 2. Types of AI

Session 3. Domains of AI

Session 1. Artificial Intelligence

Imagine you ask your mobile phone: "Hey Assistant, what's the weather today?" Here, a 3-step process is carried out by your smartphone.

Speech Recognition (Understanding): The AI first listens to your voice and converts your spoken words into text using *speech-to-text algorithms*.

Natural Language Processing (Thinking): It analyses the meaning of your question ("weather today") and figures out what you really want.

Decision Making (Reasoning): The AI searches its database or connects to a weather service to decide the correct answer.

Response Generation (Acting): Finally, it speaks back: "Today it will be sunny, with a high of 32°C." Fig 1.2 illustrates concept of AI.

In this session we are going to discuss about Artificial Intelligence (AI).

Fig. 1.2: Concept of AI.

1.0 Introduction

Artificial Intelligence (AI), usually mean computer systems that are designed to do things that normally require human intelligence. That could be something as simple as

recognizing your voice when you ask Alexa to play music, or as complex as helping doctors detect diseases early.

Human intelligence is natural intelligence. Machines such as, Computer, smartphones are not intelligent like humans.

An AI system, at its core, is made up of data, algorithms, and some level of learning. These systems use huge amounts of information, follow step-by-step instructions (that's what algorithms are), and try to make decisions or predictions based on what they've seen before.

Artificial Intelligence refers to the simulation of human intelligence in machines that are designed to think, learn, and make decisions. These systems can mimic cognitive functions such as learning from experience, understanding language, recognizing patterns, and solving problems. AI is a field within computer science that aims to build smart machines capable of performing tasks that typically require human intelligence. In the future AI will work with humans. Figure 1.3 represents collaborative working between AI and humans.

Fig 1.3: AI and Human Collaborative Working

John McCarthy, one of the founders of AI, defined it in 1956 as "the science and engineering of making intelligent machines." In simpler terms, AI allows machines to "learn" from data, adapt to new inputs, and perform human-like activities. Figure 1.4 shows a photo of John McCarthy.

Fig. 1.4: John McCarthy

The scope of AI includes everything from voice recognition in smartphones to advanced robotics in manufacturing. It is used in healthcare to assist with diagnosis, in education to personalize learning, and in finance to detect fraud. As technology evolves, AI continues to expand its role in daily life, improving efficiency, accuracy, and convenience across multiple industries.

1.0 History and Evolution of AI

The journey of AI began long before computers existed, with ancient myths about intelligent automatons. However, real progress started in the 20th century.

In 1950, British mathematician Alan Turing published a paper titled “Computing Machinery and Intelligence,” where he proposed the Turing Test, a method to determine if a machine can exhibit intelligent behaviour indistinguishable from a human. Figure 1.5 shows Alan Turing with his machine.

Fig. 1.5: Alan Turing

In 1956, during a workshop at Dartmouth College, the term "Artificial Intelligence" was officially coined. This event marked the birth of AI as an academic field. In the 1960s and 70s, researchers built simple AI programs like ELIZA, a chatbot and SHRDLU, a robot that understood basic commands.

In the 1980s, expert systems became popular, where computers could make decisions based on a knowledge base and set rules. In 1997, IBM's Deep Blue defeated world chess champion Garry Kasparov, proving that machines could compete with humans in complex games. Figure 1.6 shows IBM’s Deep Blue machine.

Fig. 1.6: IBM’s Deep Blue

The 2000s and 2010s saw the rise of machine learning and deep learning, where computers learned from large amounts of data. In 2016, Google’s AlphaGo defeated a top human player in the game of Go—a major AI milestone. Figure 1.7 shows Google’s AlphaGo.

Fig. 1.7: Google's AlphaGo

Today, AI powers voice assistants, self-driving cars, recommendation systems, and much more. Table 1.1 shows the history and evolution of AI.

Table 1.1: History and Evolution of AI

Period / Year	Key Event / Development
Ancient times	Myths and stories of intelligent automatons
1950	Alan Turing published *"Computing Machinery and Intelligence"* and proposed the Turing Test
1956	Dartmouth Workshop → Term "Artificial Intelligence" officially coined; AI became an academic field
1960s–1970s	Early AI programs developed: ELIZA (chatbot) and SHRDLU (robot understanding commands)
1980s	Rise of Expert Systems (rule-based decision-making AI)
1997	IBM's Deep Blue defeated world chess champion Garry Kasparov
2000s–2010s	Growth of Machine Learning & Deep Learning with big data
2016	Google's AlphaGo defeated top Go player → Major AI milestone
Present Day	AI is widely used in voice assistants, self-driving cars, recommendation systems, healthcare, etc.

1.2 Applications of AI in Daily Life

AI has become a part of our everyday experiences, often without us realizing it. Here are some common ways AI is used in daily life:

1. Virtual Assistants like Siri, Alexa, and Google Assistant use AI to understand and respond to spoken commands. They can set reminders, answer questions, and control smart home devices. Figure 1.8 shows symbols of Alexa, Siri and Google Assistant.

Fig. 1.8: Alexa, Siri, Google Assistant

2. Recommendation Systems are used by platforms like Netflix, YouTube, and Amazon to suggest movies, videos, or products based on user preferences and behaviour. This improves user experience and engagement. Figure 1.9 shows YouTube recommendations of similar programs.

Fig. 1.9: YouTube Recommendation System

3. Navigation and Traffic Management rely on AI to provide real-time directions and updates. Apps like Google Maps analyse traffic data to suggest the fastest routes. Figure 1.10 shows the fastest route on Google Map.

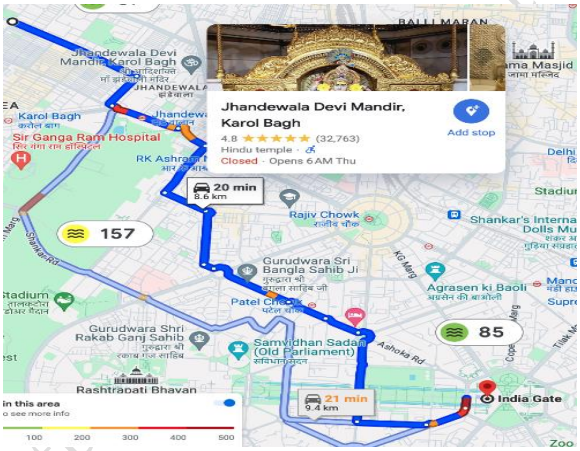


Fig. 1.10: Google Map

4. Healthcare uses AI for tasks such as disease prediction, medical image analysis, and even robotic surgery. AI can detect conditions like cancer earlier and more accurately than traditional methods. Figure 1.11 shows disease prediction processes using AI.

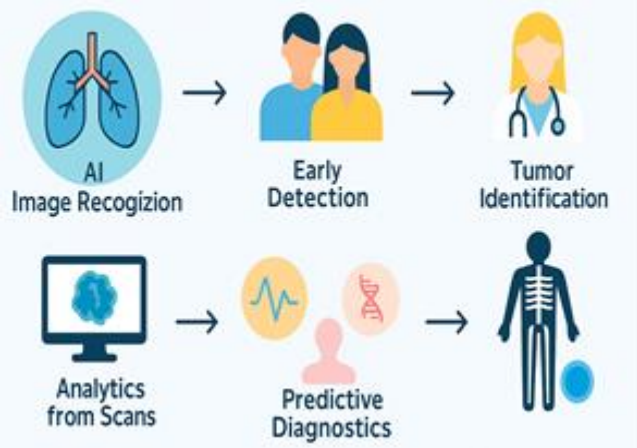


Fig. 1.11: Disease Prediction Using AI

5. Smart Homes use AI to automate lighting, temperature, and security. Devices learn from user habits to adjust settings automatically. Figure 1.12 shows automation in homes using AI.

Fig. 1.12: Smart Home Using AI.

6. Banking and Finance industries use AI for fraud detection, credit scoring, and customer service through chatbots. Figure 1.13 shows SBI Chatbot for customer service.

Fig. 1.13: SBI Chatbot for customer service

AI makes technology smarter, more efficient, and responsive to individual needs.

1.3 AI in School Education

In School Education, we normally teach those subjects that are useful for the students to make their career in that subject. Also, we teach those subjects that are frequently needed when he faces different real-life problems in the real world. The subject of artificial intelligence is developed over a long period of time of about 75 years. The use of AI in human lives has increased by many folds. Many people knowingly or unknowingly are making use of AI. Hence it is necessary that a student studying at school in class 11 should be aware of this topic. The course on AI can significantly enhance school education in the following ways. Figure 1.14 shows AI in School Education.

Fig. 1.14: AI in School Education

1. Professional development: professionals in computer science or AI will be developed at an early stage.

2. Ability of data analysis: with AI, a student can easily analyse the data to obtain fruitful conclusions.

3. Accessibility: with AI a student can access many resources. Even students with disabilities can make use of AI tools.

4. Intelligent learning: a student can learn the subject even outside the classroom with the help of an intelligent learning environment.

1.4 Role and Responsibilities of AI Associate Personnel

An AI Associate is a professional who assists in the development, testing, deployment, and maintenance of artificial intelligence systems. They may not build AI systems from scratch like data scientists or machine learning engineers, but they play a key supportive and operational role in AI-related projects.

In simpler terms, think of them as the team players who understand how AI works and help make it run smoothly in real-world situations.

An AI Associate works under the guidance of senior AI professionals and performs a variety of tasks that involve data, tools, and AI workflows.

1. Data Handling and Preparation

1. Collecting and organizing datasets for training and testing AI models.
2. Cleaning data to remove errors, duplicates, or irrelevant information.
3. Labelling data for supervised learning (e.g., tagging images or text).

2. Model Testing and Evaluation

1. Running tests on AI models to check their performance.
2. Reporting issues like incorrect predictions or model bias.
3. Helping fine-tune models to make them more accurate.

3. Tool and Platform Usage

1. Using tools like Teachable Machine, TensorFlow, IBM Watson, or Microsoft Azure AI for AI model development and deployment.
2. Assisting in integrating AI tools with websites, apps, or software systems.

4. Documentation and Reporting

a) Keeping track of project progress and documenting results.
b) Preparing simple reports to explain findings or AI performance to team members or clients.

5. Ethical Awareness

I. Following AI ethics guidelines such as ensuring data privacy and reducing bias.
II. Reporting any misuse or potential harm caused by AI systems.

6. Team Collaboration

1. Working with data scientists, developers, and project managers.
2. Participating in team meetings and helping to brainstorm AI solutions for business problems.

Knowledge Required for an AI Associate

To be successful in this role, one must have a basic understanding of AI and its related concepts.

Fundamental Knowledge

(i) Basics of AI and Machine Learning
(ii) Understanding of data types: images, text, numbers, etc.
(iii) Familiarity with AI terms like algorithms, models, training, and prediction.
(iv) Knowledge of AI applications in industries like healthcare, education, and finance.

Awareness of AI Ethics

1. Understanding the importance of fairness, transparency, and privacy in AI.
2. Ability to identify bias in data or models.

Table 1.2 shows the role and responsibilities of AI associate personnel.

Table 1.2: Role and Responsibilities of AI Associate Personnel

Aspect	Details
Role	Assist in AI model development, data handling, and testing.
Responsibilities	Data preparation, testing, documentation, ethical usage, teamwork.
Knowledge	Basics of AI, ML, data, and ethical AI.
Skills	Programming (basic), communication, teamwork, use of AI tools.

Skills Required for an AI Associate

1. Technical Skills:

1. *Basic Programming:* Knowledge of Python is helpful.
2. *Data Handling Tools:* Working with Excel, Google Sheets, or simple databases.
3. *AI Platforms:* Using tools like Google Teachable Machine or Scratch with AI extensions.

2. Analytical Thinking:

1. Ability to solve problems logically.
2. Spotting patterns and understanding AI predictions.

3. Communication Skills:

a) Explaining AI concepts in simple terms to non-technical people.
b) Writing clear and simple reports.

4. Teamwork and Collaboration:

1. Working effectively in a team.
2. Open to feedback and new ideas.

5. Adaptability:

1. Willingness to learn new AI tools and techniques.
2. Keeping up with fast-changing AI trends and technologies.

Being an AI Associate is a great starting point for students interested in artificial intelligence. It opens the door to exciting careers in AI, data science, and robotics. With basic knowledge, hands-on practice, and curiosity, students can become valuable contributors to the AI-powered future.

1.5 Framework of AI Associate

In Artificial Intelligence, a framework refers to a set of tools, libraries, and services that help build, train, and deploy AI models or applications. For an AI Associate, frameworks make it easier to create conversational AI, such as chatbots, voice assistants, and smart applications—without needing deep coding knowledge.

Let's explore three popular AI frameworks:

1. Google Dialog Flow

Google Dialog Flow is a cloud-based tool that helps developers (and even beginners) build chatbots and voice assistants that understand natural language. Figure 1.15 shows Google Dialog Flow.

Fig. 1.15: Google Dialog Flow

Features:

(i) Supports both text and voice-based conversations.
(ii) Works with Google Assistant, websites, mobile apps, and social media.
(iii) Uses intents (user goals) and entities (specific data) to understand input.
(iv) Supports multiple languages.

Role of an AI Associate:

1. Create conversational flows (intents and responses).
2. Test chatbot interactions.
3. Train the model using sample phrases.
4. Deploy the assistant on websites or Google Assistant.

Example: A student can use Dialog Slow to build a school inquiry chatbot.

2. IBM Watson Assistant

IBM Watson Assistant is a powerful AI platform that enables users to build and deploy AI-powered chatbots and virtual agents for websites, mobile apps, and messaging platforms. Figure 1.16 shows IBM Watson Assistant.

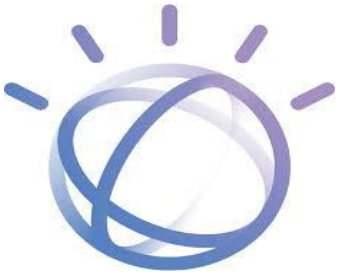


Fig. 1.16: IBM Watson Assistant

Features:

I. Uses natural language understanding (NLU).
II. Offers drag-and-drop tools to design chat flows easily.
III. Integrates with Slack, Facebook Messenger, WhatsApp, and more.
IV. Can connect to back-end systems to provide real-time responses.

Role of an AI Associate:

I. Design dialogue trees and responses.
II. Connect Watson to a company database or website.
III. Use pre-built models or customize intents.
IV. Monitor and improve chatbot accuracy.

Example: A Watson Assistant chatbot can answer customer service queries for a business.

3. Amazon Alexa Skills Kit (ASK)

The Amazon Alexa Skills Kit (ASK) is a set of tools that allows developers to create voice-based apps, called Alexa Skills, for Amazon Echo and other Alexa-enabled devices. Figure 1.17 shows ASK.

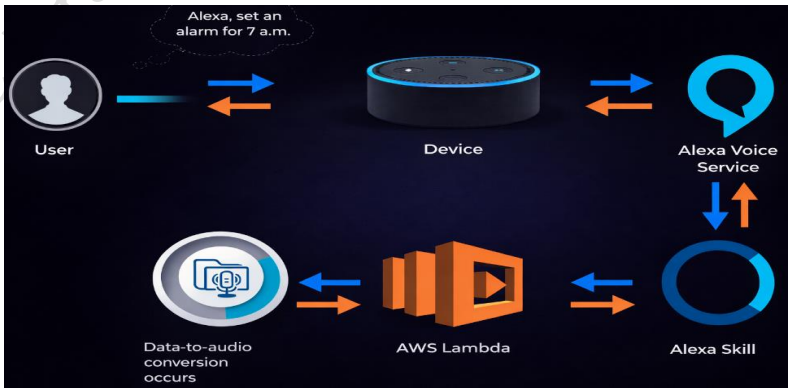


Fig. 1.17: ASK

Features:

1. Enables voice interaction through "Skills" (like apps for Alexa).
2. Supports custom commands, smart home integration, and games.

3. Uses voice intents and utterances to recognize what users say.
4. Integrates with smart devices (IoT), music, and shopping services.

Role of an AI Associate:

1. Build and test simple voice commands (skills).
2. Use Alexa developer console to design voice responses.
3. Train Alexa with example user phrases.
4. Connect Alexa skills to APIs or databases.

Example: An Alexa skill can tell students the daily timetable or provide math quiz games.

Element	**Details**
Google Dialog Flow	Great for building text/voice chatbots using Google Assistant.
IBM Watson Assistant	Ideal for professional chatbots with business applications.
Amazon Alexa Skills Kit	Best for voice-based AI apps using Alexa.
AI Associate Role	Designing flows, training intents, testing, and deploying assistants.

These AI frameworks are powerful tools that can be used by students and beginners to create amazing voice and chat-based applications. As an AI Associate, learning how to use these platforms gives you a solid foundation for advanced AI careers in the future.

1.6 Future Scope of AI Associate

As Artificial Intelligence becomes more integrated into daily life, the demand for skilled AI professionals is increasing rapidly. Starting as an AI Associate can be the first step toward a wide range of high-growth, well-paying careers in the AI ecosystem as given in the following table.

Aspect	**Future Scope**
Career Growth	AI Developer, Data Scientist, ML Engineer
Industries	Healthcare, Retail, Finance, Agriculture, Education
Further Learning	Certifications, Degrees, Online Courses
Job Demand	High and growing globally
Other Paths	Freelancing, start-ups, consulting

Summary

- AI simulates human intelligence in machines using data, algorithms, and learning.
- History: From Alan Turing (1950, Turing Test) → Dartmouth Workshop (1956) → Expert Systems (1980s) → Deep Blue (1997) → AlphaGo (2016).
- Applications: Virtual assistants, recommendation systems, navigation, healthcare, smart homes, banking.

- AI in education: improves learning, accessibility, and professional skills.
- Role of AI Associate: handling data, testing models, using tools, ensuring ethics, and collaborating in teams.
- Frameworks: Google Dialogflow, IBM Watson, Amazon Alexa Skills Kit.
- Future scope: careers in AI, ML, data science across industries.

Check your progress

A. Multiple Choice Questions

1. Who coined the term "Artificial Intelligence" in 1956? (a) Alan Turing (b) John McCarthy (c) Marvin Minsky (d) Garry Kasparov
2. Which test was proposed by Alan Turing in 1950? (a) CAPTCHA (b) Turing Test (c) Deep Blue Test (d) Loebner Test
3. Which AI program is considered the first chatbot? (a) MYCIN (b) ELIZA (c) SHRDLU (d) Watson
4. IBM's Deep Blue defeated Garry Kasparov in: (a) 1980 (b) 1997 (c) 2005 (d) 2016
5. Which of the following is NOT an AI application? (a) Google Maps (b) Siri (c) Netflix recommendations (d) Mechanical calculator

B. Fill in the blanks

1. AI refers to the simulation of _____ intelligence in machines.
2. The chatbot developed in the 1960s was called _____.
3. IBM's _____ defeated world chess champion Garry Kasparov.
4. In 2016, Google's _____ defeated a top Go player.
5. AI systems are mainly built upon data, _____, and learning.

C. True or False

1. Human intelligence and machine intelligence are the same.
2. AI can be used in healthcare for early disease detection.
3. AI is only used in robotics and not in finance.
4. The scope of AI includes education, finance, and smart homes.
5. AI always makes correct decisions without error.

D. Short Answer Questions

1. Define Artificial Intelligence.
2. Write two applications of AI in daily life.
3. Who is considered the father of AI?
4. What are the three key components of an AI system?
5. Mention one role of an AI Associate.

Session 2. Types of AI

Do you know AI-powered Social Robots like Sophia by Hanson Robotics. It tries to understand human emotions, intentions, and social interactions. If you smile, it can recognize happiness; if you frown, it adjusts responses. It is still evolving — the goal is machines that "understand" human feelings like humans do. Figure 2.1 shows AI powered social robots.

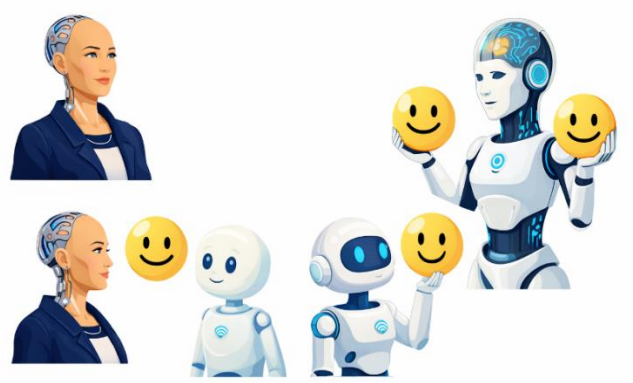


Fig 2.1: AI powered Social Robot.

Artificial Intelligence (AI) can be divided into different types based on its capabilities and functionalities. These types help us understand how advanced or intelligent an AI system is and what it can do.

Just like humans go through stages of learning—like a baby, a student, and a scientist—AI also has levels of intelligence, from simple to super smart.

In this session we will discuss different types of AI.

2.1 Types of Artificial Intelligence Systems Based on Capabilities

AI is classified based on its capabilities. In other words, how “intelligent” the system actually is.

There are three main types in this category: **ANI, AGI, ASI**

1. Narrow AI (ANI), what we have today
2. General AI (AGI), what researchers are trying to build
3. Super AI (ASI), what we imagine might exist in the future

Figure 2.2 shows three types of AI.

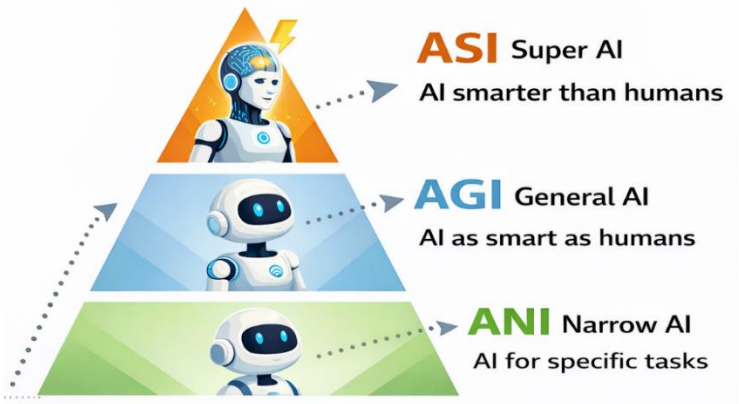


Fig. 2.2: Types of AI

1. Artificial Narrow Intelligence (ANI):

This is also known as Weak AI. ANI is designed to perform a single task or a narrow range of tasks. It doesn't think like a human, it doesn't "understand" anything the way we do, and it definitely can't switch from one task to another. It cannot perform beyond what it is programmed to do. Most AI applications today fall under ANI such as, Alexa, Navigation Map, ChatGPT and recommendation system as shown in Figure 2.3.

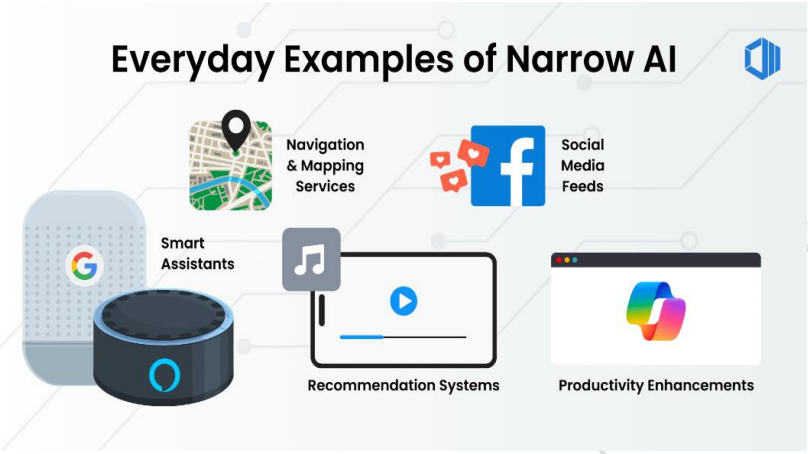


Fig. 2.3: Narrow AI Examples

Real-World Examples of Narrow AI:

ChatGPT: It can chat with you, write emails, and explain stuff, but that's it. It doesn't "know" anything in the human sense.

Siri and Alexa: These voice assistants can answer questions, perform actions based on voice commands, set timers, and play songs, but they don't actually understand the world like you or me.

Google Maps: It finds the fastest route using traffic data. Super helpful. But it won't help you write a story.

Netflix and Spotify: They suggest shows or songs you might like, based on what you've already watched or listened to.

Spam Filters in email that automatically sort or delete unwanted messages. Figure 2.4 shows the working of spam filters.

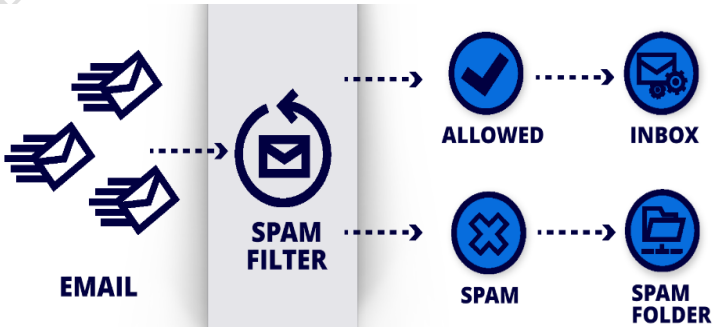


Fig 2.4: Working of Spam filter

Autopilot systems in airplanes and cars that assist with navigation and driving as shown in Figure 2.5.

Fig. 2.5: Autopilot in Tesla Car

Although ANI can perform its tasks extremely well, it lacks the ability to think or understand concepts outside of its programming. It cannot transfer knowledge from one task to another.

2. Artificial General Intelligence (AGI):

AGI refers to a system with general cognitive abilities. In theory, AGI would be able to understand and perform any intellectual task that a human can.

Unlike ANI, which is limited to one domain, AGI would be able to switch between tasks, learn from experience, and adapt to new situations without being explicitly programmed for each task. For example, an AGI system could teach itself to play chess, learn a new language, and even compose music—all without human help. Figure 2.6 shows the prototype AGI image.

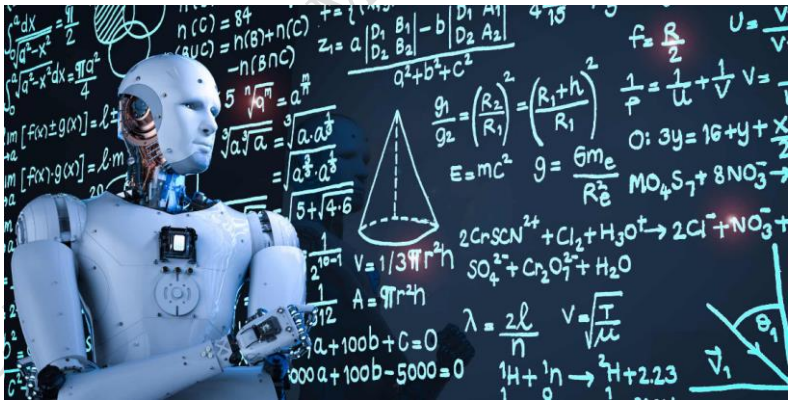


Fig. 2.6: AGI

Currently, AGI does not exist. It remains a goal for researchers, and building such systems requires significant advances in computing, neuroscience, and ethics. It is also called Strong AI, AGI can understand, learn, and apply knowledge across a wide range of tasks just like a human being. It could solve math problems, write poetry, learn new skills, and even understand jokes or emotions. Basically, it would think and learn like a person.

The tools like ChatGPT and Gemini are amazing at generating human-like responses. But they're still Narrow AI. They don't actually "understand" the conversation, they're just really good at predicting the next word.

Figure 2.7 shows capabilities of General AI such as visual perception, audio perception, fine motor skills, natural language processing, problem solving, navigation, creativity and social and emotional engagement.

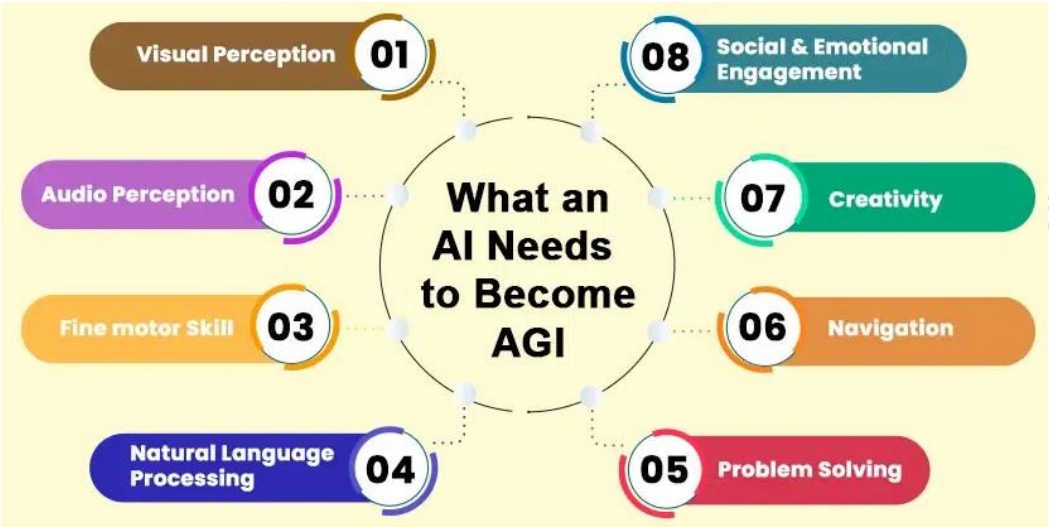


Fig. 2.7: Capabilities of General AI

3. Artificial Super Intelligence (ASI):

ASI goes beyond human intelligence. It would be smarter than humans in every possible way. This is a hypothetical form of AI that would surpass human intelligence in every aspect—from creativity and decision-making to emotional intelligence. While ASI is still science fiction, it is often discussed in philosophical and ethical debates.

Figure 2.8 shows prototype Super AI.

Fig. 2.8: Super AI

This type of AI is purely theoretical and has not yet been developed. However, it raises important ethical concerns such as, Will ASI act in humanity's best interest? Can we control something smarter than ourselves? Who is responsible if it causes harm?

These questions have sparked debate among scientists, ethicists, and technologists. Ensuring that such intelligence is safe and aligned with human values is one of the major concerns of future AI development.

ASI could solve climate change, invent a new language, write the next best-selling novel, and negotiate world peace.

Figure 2.9 shows capabilities of Super AI such as consciousness, beliefs, desires, cognition, emotional intelligence, subjective experience and behavioural intelligence.

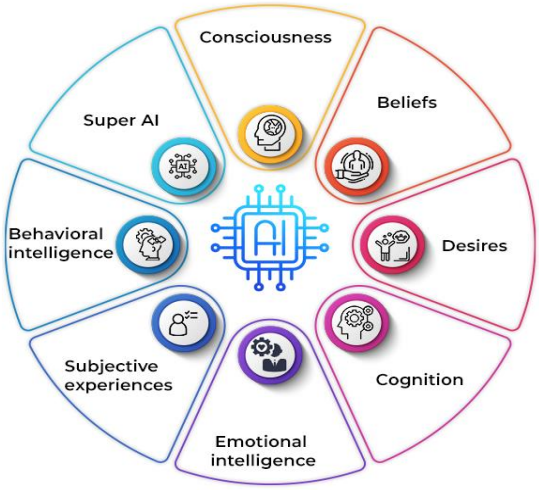


Fig. 2.9: Capabilities of Super AI

Let's summarise the types of AI as,

Type	Exists Now?	What It Can Do	Example(s)
Narrow AI	Yes	One task really well	ChatGPT, Netflix, Google Maps
General AI	Not yet	Think and learn like a human	Still a theory
Super AI	Nope	Smarter than all humans combined	Total speculation (for now)

2.2 Types of AI Systems Based on Functionality

Another way to classify AI is by how the system behaves, what it can actually do in the moment, and how it responds to information. This classification focuses on whether the AI can learn from experience, remember past events, or understand people emotionally.

There are four main types of AI under this category: *Reactive Machines, Limited Memory, Theory of Mind, Self-aware AI*

1. Reactive Machines

This is the simplest kind of AI. Reactive machines don't remember anything. They just respond to the situation in front of them. It gives output based on input, no learning, no context, no memory of past events. Figure 2.10 shows reactive AI machine structure.

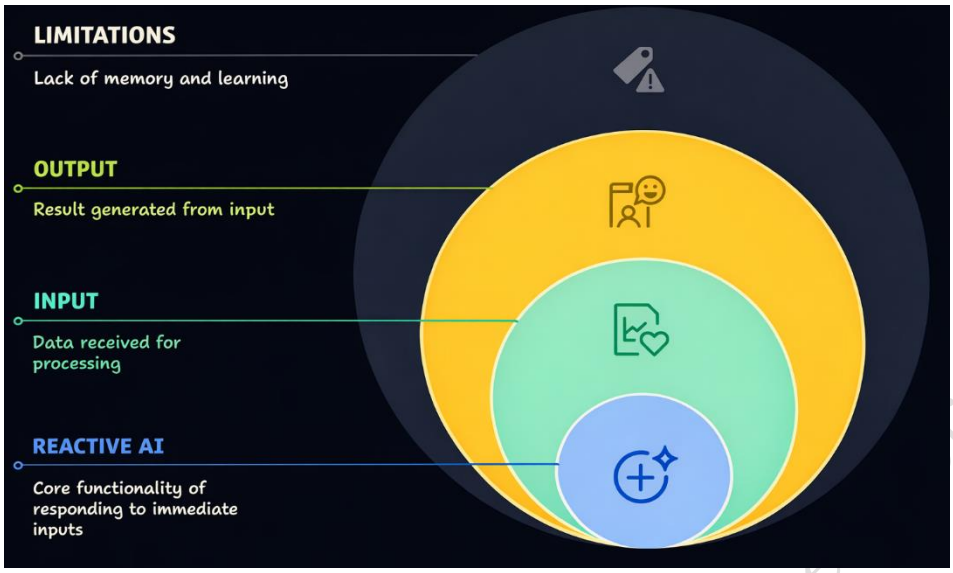


Fig. 2.10: Reactive AI machine structure

Reactive machines are extremely focused. They don't get confused, but they also don't improve or adapt. Think of them like a really smart calculator: precise, fast, but totally unaware of anything beyond the numbers.

IBM's Deep Blue is a classic example of the chess computer that beat Garry Kasparov in the '90s. It could evaluate a huge number of moves and pick the best one, but it had no memory of previous games or personal strategy. Every move was based on the current board only.

2. Limited Memory

Now we're getting into AI that can *learn*, at least a little. Limited memory systems can use **past data to inform current decisions**. That means they can get better over time. They still don't "understand" like humans do, but they can notice patterns and make adjustments.

Consider an example of Self-driving cars. They observe road conditions, nearby vehicles, speed, and traffic signals. Based on all that, they make real-time driving decisions based on the data to achieve better accuracy for staying safe on the road. Due to limited memory, these systems don't have long-term recall or a sense of personal history. They just remember short-term data relevant to their task.

3. Theory of Mind (In Development)

The researchers are working on Theory of Mind. It would be able to understand people, recognizing emotions, beliefs, intentions, and even subtle social cues.

If AI could truly understand what a user *feels* or *wants*, it could become much better at things like healthcare, therapy, education, and customer service. Figure 2.11 shows AI's Theory of Mind.

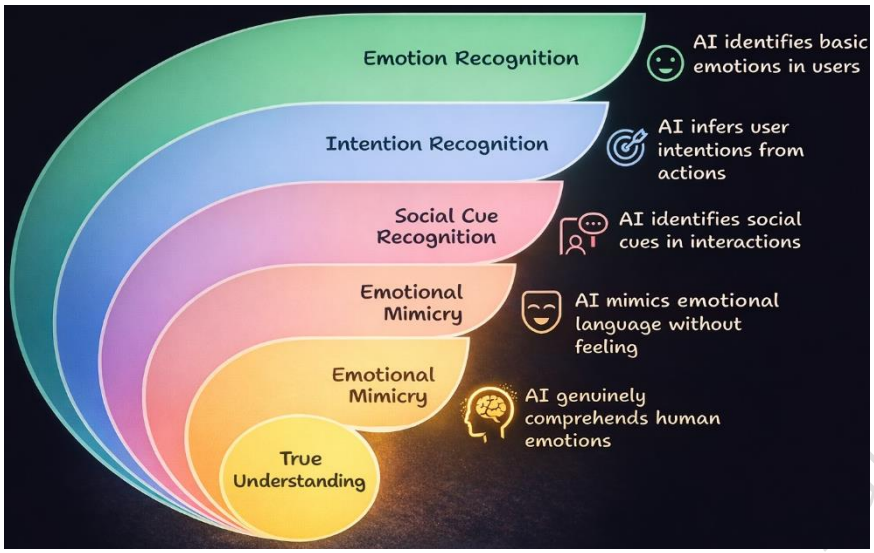


Fig. 2.11: AI's Theory of Mind

Right now, even the most advanced AI doesn't really "get" human emotions. It can mimic emotional language, sure, but it doesn't *feel* anything or understand what it's like to be frustrated, excited, or scared.

4. Self-aware AI (Purely Hypothetical)

This is the furthest point on the AI roadmap, and it's still a science fiction idea. A self-aware AI would not only understand emotions, it would have its **own consciousness**. It would know that it exists, understand itself, and possibly even have its own goals.

No such AI exists today. There's not even a clear path to building one. It's more of a philosophical concept at this point, but it often shows up in movies, like *Ex Machina* or *Her*.

If we ever got to this stage, the ethical questions would be enormous. Could a self-aware AI have rights? Could it be held accountable for its actions? These are tough questions we might face someday, but not today.

Types of AI Systems Based on Learning Techniques

AI isn't just about being smart, it's also about **how it gets smart**. That's where machine learning comes in. There are a few different ways that AI systems learn from data. These learning styles shape what the system can do, how flexible it is, and where it can be used.

1. Supervised Learning

This is the most common method. In supervised learning, the AI is trained on labelled data, meaning the input and correct output are both provided during training.

Imagine giving the system a thousand emails and telling it which ones are spam. It looks at the examples and figures out the patterns that separate spam from regular messages. Then it uses that learning to predict spam in the future.

Supervised learning is used in tons of places:

- Email spam filters
- Loan approval systems

- Facial recognition
- Medical image classification

It's great when you have a lot of clear, clean training data. But if the labels are messy or biased, the system can pick up those same problems.

2. Unsupervised Learning

In this setup, the AI is given data, but no labels. It has to find patterns or groupings on its own.

One of the most common uses is customer segmentation. If you run an online store, unsupervised learning can look at all your customer behaviour and break it into groups, maybe frequent buyers, discount shoppers, or browsers who never check out.

The AI isn't told what each group means. It just notices the similarities and clusters them.

Unsupervised learning is useful when you want insights from raw data without already knowing what you're looking for.

3. Reinforcement Learning

This is more like how animals (and humans) learn. The system interacts with an environment, tries different actions, and gets rewarded or punished depending on how well it does. Think of it like training a dog, it does something right, it gets a treat. It does something wrong, no threat.

AI systems like AlphaGo, which beat the world's best Go players, use reinforcement learning. So, do many robotics systems and even some recommendation engines.

It's powerful for tasks that involve lots of trial and error, especially when the environment is constantly changing.

4. Deep Learning

This is a special kind of machine learning that uses neural networks, computer systems inspired by how the human brain works.

Deep learning is what powers things like:

- Voice assistants (like Siri and Alexa)
- Image recognition (like your phone's Face ID)
- Natural language tools (like ChatGPT or Google Translate)

The "deep" part refers to multiple layers of these artificial neurons. The more layers, the more complex the patterns the AI can recognize.

Deep learning has been responsible for many of the big AI breakthroughs we've seen in the past 10 years. It's especially good at handling unstructured data, stuff like photos, audio, video, and natural language.

Types of AI Systems by Application Area

AI isn't just one big system doing the same thing everywhere. In the real world, it shows up in different forms depending on where it's being used. From your Netflix account to a robot surgeon in a hospital, AI is being shaped to fit the job it's supposed to do.

Let's break down the types based on where they're actually used, not in theory, but in real life.

1. Expert Systems

These are like digital advisors. Expert systems are trained using rules and knowledge from professionals in a specific field. They don't guess or come up with creative ideas, they follow logical steps and make decisions based on that.

You'll see them in things like:

- Medical diagnosis tools that suggest what illness a patient might have
- Financial apps that give investment recommendations
- Legal tech that helps review contracts for issues

They're fast, consistent, and pretty reliable, as long as they're built well and fed good info.

2. NLP (Natural Language Processing) Systems

This is the kind of AI that deals with language, understanding it, analyzing it, and even generating it. If you've ever asked Siri a question or chatted with a bot on a website, you've used an NLP system.

These are used for:

- Translating languages
- Writing text or summaries
- Customer service chatbots
- Voice recognition apps

They're not perfect, they might misinterpret what you say, but they're getting smarter by the day.

3. Computer Vision Systems

This one's all about helping machines "see." Computer vision systems look at images or video and try to understand what's going on.

Real-life uses include:

- Face unlocks on your phone
- Detecting objects or people in security cameras
- Self-driving cars recognizing road signs or lane lines
- Diagnosing diseases from medical scans

These systems have been trained on millions of images, so they're surprisingly good at recognizing things. Sometimes better than humans, especially in areas like medical imaging.

4. Robotics Systems

When you combine AI with machines that can move and interact with the physical world, you get robotics.

Robotic AI is used in:

- Factories (automated arms on assembly lines)
- Delivery robots
- Surgical robots that help doctors during procedures

- Humanoid robots that can walk, climb stairs, or even dance

Some robots follow strict instructions, but the more advanced ones are using AI to make real-time decisions, especially in unpredictable environments.

5. Recommender Systems

These are the engines behind why you see *that* song, *that* movie, or *that* product at the perfect time.

Recommender systems learn from your behaviour, what you clicked, watched, or liked, and then try to guess what you'll enjoy next.

The power:

- Netflix suggestions
- Spotify playlists
- YouTube video queues
- E-commerce product recommendations

It's a mix of smart math and user behaviour, not magic, just very good guessing based on patterns.

6. Predictive Analytics Systems

This type of AI doesn't just react, it tries to predict what might happen next.

It's used for:

- Forecasting sales
- Predicting customer behaviour (like who might cancel a subscription)
- Risk scoring for loans or insurance
- Even predicting when a machine might break down in a factory

Companies love predictive AI because it helps them plan better and avoid problems before they happen.

1.5 Branches of AI

AI is a broad field made up of several specialized branches:

a) ***Natural Language Processing (NLP):*** Enables machines to understand and generate human language.

b) ***Computer Vision (CV)***: Allows machines to interpret and process visual data.

c) ***Robotics: C***ombines AI with mechanical systems to build intelligent robots.

d) ***Expert Systems:*** Mimic human experts in specific fields to solve complex problems.

e) ***Machine Learning (ML):*** A subfield that teaches machines to learn from data and improve over time.

Each of these branches contributes to the development of smarter systems across various industries.

1.6 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a branch of AI that helps machines understand, interpret, and respond to human language in a way that is meaningful. It bridges the gap between human communication and computer understanding.

NLP Working

NLP involves breaking down language into parts such as words, sentences, and meanings. It uses grammar rules and data patterns to understand what people are saying or writing. Advanced NLP also includes understanding the context and emotions behind words. Fig 2.12 shows how NLP works. Human audio is captured and it is converted into text. The text data is processed by machine for response. Response is converted into audio back.

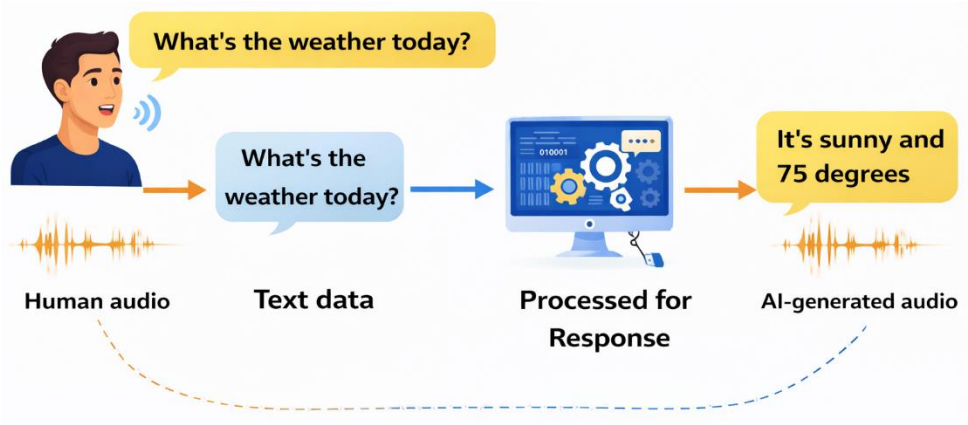


Fig. 2.12: NLP working

Applications of NLP

1. ***Chatbots:*** AI chatbots use NLP to answer customer queries.
2. ***Translation Tools:*** Google Translate uses NLP to convert one language to another.
3. ***Voice Assistants:*** Siri and Alexa understand and respond to spoken commands using NLP.
4. ***Sentiment Analysis:*** Companies use NLP to understand customer opinions in reviews and social media.

NLP is an essential part of how we interact with smart devices using everyday language.

1.7 Computer Vision (CV)

Computer Vision (CV) is a field of AI that enables machines to see, interpret, and respond to visual information such as images and videos.

Working of CV

Computer Vision systems use cameras, sensors, and algorithms to capture and analyse visual input. They can detect shapes, objects, colours, faces, and even movements. Deep learning models help in recognizing complex patterns in visual data. Fig 2.13 shows face recognition process.

Applications of Computer Vision

1. **Facial Recognition:** Used in phone unlocking and surveillance.
2. **Medical Imaging:** Helps detect diseases by analysing X-rays, MRIs, and CT scans.
3. **Autonomous Vehicles:** Self-driving cars use CV to detect road signs, obstacles, and pedestrians.
4. **Retail:** AI can track customer movement in stores for better layout and design.

Fig. 2.13: working of computer vision

Computer Vision allows machines to process and "understand" what they see, just like humans do.

1.8 Robotics

Robotics is a branch of AI that deals with designing and building robots—machines capable of performing tasks with or without human intervention.

AI in Robotics

AI gives robots the ability to:

1. ***Sense:*** Use sensors to perceive the environment.
2. ***Think*:** Make decisions using AI algorithms.
3. ***Act*:** Perform tasks like moving, lifting, or assembling.

Types of Robots:

1. ***Industrial Robots:*** Used in factories for manufacturing and assembly. Fig 2.14 shows industrial robots.

Fig. 2.14: Industrial Robots

2. ***Service Robots:*** Help with household tasks, customer service, or cleaning.
3. ***Medical Robots:*** Assist doctors in surgeries or rehabilitation.

4. ***Exploration Robots:*** Used in dangerous environments like space (e.g., Mars rovers).

Robotics combines hardware and AI to create machines that can perform complex actions safely and efficiently.

1.9 Expert Systems

An Expert System is an AI program that simulates the decision-making ability of a human expert. It uses a set of rules and a knowledge base to solve problems in specific fields like medicine, law, or engineering. Fig 2.15 shows block diagram of expert system.

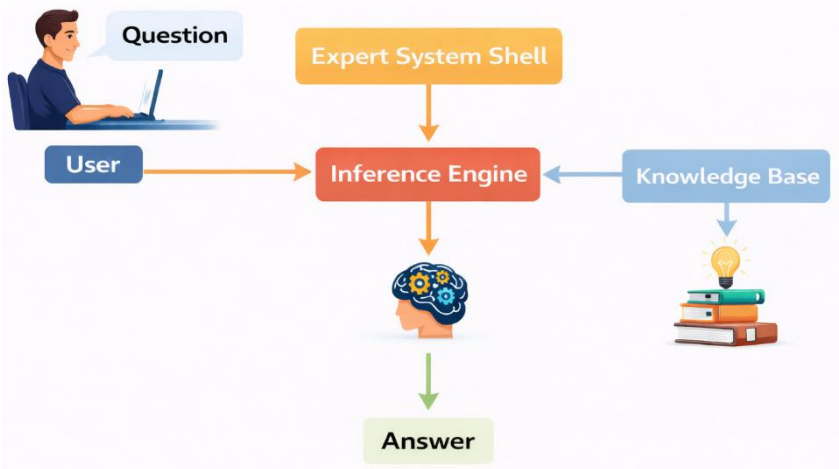


Fig. 2.15: Block Diagram of Expert System

Main Components

1. ***Knowledge Base:*** A collection of facts and rules related to a particular subject.
2. ***Inference Engine:*** The logic that applies the rules to known facts to reach conclusions.
3. ***User Interface:*** It allows users to interact with systems.

Applications of Expert Systems

1. ***Medical Diagnosis:*** Systems like MYCIN help doctors diagnose diseases. Fig 2.16 shows the workings of MYCIN.

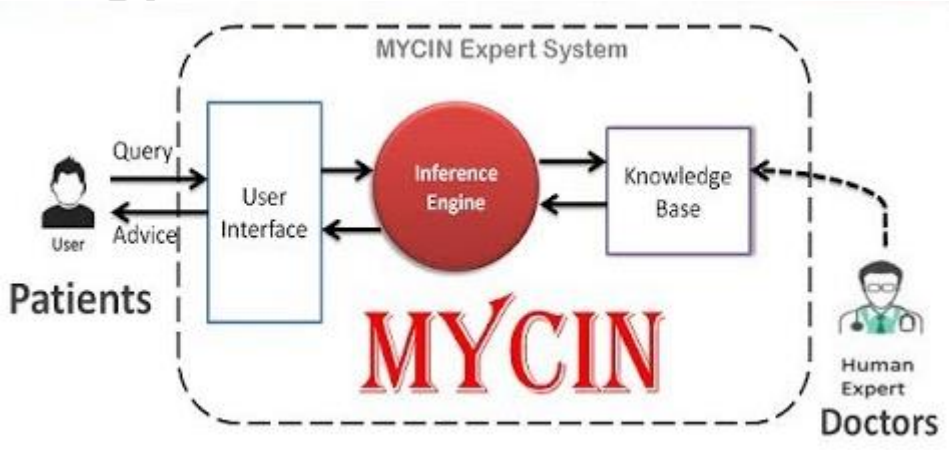


Fig. 2.16: MYCIN

2. ***Technical Support:*** Troubleshooting software for machines and computers.

3. ***Legal Advice:*** AI systems help interpret laws and legal documents.

Expert systems are useful in situations where quick, consistent decision-making is needed based on a fixed set of rules.

1.10 Ethics in AI

As AI becomes more powerful, ethical issues become more important. Ethics in AI refers to the set of principles that guide the development and use of AI systems.

Key Ethical Concerns

a) ***Fairness:*** AI should not discriminate based on race, gender, or other personal attributes.

b) ***Privacy:*** AI systems must protect personal data and avoid unauthorized surveillance.

c) ***Accountability:*** It should be clear who is responsible when AI makes a mistake.

d) ***Transparency:*** Users should understand how AI makes decisions.

e) ***Human Control:*** AI should not operate without human oversight in sensitive areas like healthcare or defence.

Ethical AI is necessary to ensure that technology serves all people fairly and safely.

1.11 Bias in AI

Bias in AI happens when an AI system makes unfair or incorrect decisions because of biased data or programming. This can lead to discrimination and inequality in AI-based decisions.

Causes of AI Bias:

1. ***Training Data Bias:*** If the data used to train the AI contains biases, the AI will learn those biases.
2. ***Human Bias:*** If developers unknowingly introduce their own biases into the system.
3. ***Sampling Bias:*** Using unbalanced datasets that don't represent all groups equally.

Real-World Examples:

1. ***Facial Recognition:*** Often less accurate for darker-skinned individuals.
2. ***Hiring Tools:*** Some AI tools favoured male resumes over female ones due to biased data.
3. ***Loan Approval:*** AI systems denying loans based on zip codes associated with certain races.

To reduce bias, developers must use diverse datasets and regularly test AI for fairness.

1.12 AI Myths vs Reality

Myth 1: AI will take over all jobs.

Reality: AI will automate some repetitive tasks but also create new job opportunities in AI development, data science, ethics, and more. Humans will still be needed for creativity, emotional intelligence, and complex decision-making.

Myth 2: AI can think like a human.

Reality: AI can simulate human-like behaviour, but it doesn't "think" or "feel" like humans. It does not have emotions, self-awareness, or consciousness. It follows patterns based on data.

Myth 3: AI is always correct.

Reality: AI systems can make mistakes, especially if the input data is poor or biased. AI should be used as a tool to assist humans, not replace human judgment completely.

Myth 4: AI is only for tech experts.

Reality: Many AI tools today are user-friendly and don't require programming knowledge. Platforms like Teachable Machine and Scratch AI extensions allow beginners and students to experiment with AI.

Understanding what AI is—and what it is not—is important to avoid fear and use it responsibly.

1.13 AI Platforms for Students

AI is not just for professionals—students can also learn and create with AI using beginner-friendly platforms:

1. ***Teachable Machine:*** A no-code tool by Google that lets you train models using your webcam or microphone.
2. ***MIT App Inventor:*** Lets students build mobile apps that can use AI features like speech recognition.
3. ***Scratch with AI Extensions:*** Allows young learners to add basic AI features to their visual coding projects.
4. ***Kaggle:*** A platform where students can access datasets, participate in challenges, and learn AI/ML by doing.
5. ***Mumbai IIT Tool Box:*** Various AI tools are available on Mumbai IIT tool box.

These tools help students experiment and understand AI concepts in a fun, hands-on way.

1.14 How to Start Learning AI

If you're interested in learning AI, here's how to begin:

1. ***Learn Basic Programming:*** Python is the most popular language for AI. Start with simple coding exercises.
2. ***Understand Math Concepts:*** Focus on logic, statistics, and algebra.
3. ***Watch Tutorials and Read Blogs:*** Use YouTube, free courses on Swayam, NPTEL, Coursera or edX to learn at your own pace.
4. ***Do Projects:*** Try simple projects like chatbot creation, number prediction, or image recognition.
5. ***Join Communities:*** Participate in AI clubs, school competitions, and online forums to collaborate and grow.

The key is to start small, stay curious, and keep learning.

AI is no longer a thing of the future—it's already shaping our present. From helping doctors diagnose diseases to recommending the next video on YouTube, AI is everywhere.

For students, understanding how AI works and how to use it responsibly is an important life skill.

Note the following:

1. AI mimics human intelligence and is used in various fields.
2. There are three types of AI: ANI, AGI, and ASI.
3. AI applications include NLP, CV, robotics, and expert systems.
4. Ethical use of AI is important to ensure fairness and safety.
5. Myths about AI often exaggerate its capabilities or dangers.
6. With the right tools and interest, anyone can begin learning AI.
7. The future of AI is exciting—and you can be a part of it.

Summary

- AI by Capabilities: ANI (Narrow AI): Task-specific (e.g., ChatGPT, Netflix), AGI (General AI): Human-level intelligence (still theoretical), ASI (Super AI): Beyond human intelligence (hypothetical).
- AI by Functionality: Reactive Machines (e.g., Deep Blue), Limited Memory (e.g., self-driving cars), Theory of Mind (in development), Self-aware AI (hypothetical).
- AI by Learning Techniques: Supervised, Unsupervised, Reinforcement, and Deep Learning.
- AI by Application Area: Expert Systems, NLP, Computer Vision, Robotics, Recommender Systems, Predictive Analytics.
- Branches of AI: NLP, CV, Robotics, Expert Systems, ML.
- Ethics & Bias: fairness, privacy, accountability, transparency.
- Myths vs. Reality: AI won't take all jobs, not conscious, not always correct.
- AI platforms for students: Teachable Machine, Scratch, Kaggle.
- How to start learning AI: Python, math basics, projects, online resources.

Check Your Progress

A. Multiple Choice Questions

1. Which type of AI exists today? (a) AGI (b) ASI (c) ANI (d) Self-aware AI
2. Which type of AI can learn, adapt, and switch between tasks like a human? (a) ANI (b) AGI (c) ASI (d) Reactive Machines
3. Which AI beat the world champion in Go in 2016? (a) Deep Blue (b) Watson (c) AlphaGo (d) DENDRAL
4. Self-driving cars use which type of AI functionality? (a) Reactive Machines (b) Limited Memory (c) Theory of Mind (d) Self-aware AI
5. Which learning type uses labelled data? (a) Unsupervised Learning (b) Supervised Learning (c) Reinforcement Learning (d) Deep Learning

B. Fill in the blanks

1. AI based on capabilities is classified into ANI, AGI, and _____.
2. _____ Machines do not use memory of past events.
3. In _____ learning, AI finds hidden patterns without labelled data.
4. _____ learning is based on rewards and punishments.
5. Deep learning uses artificial _____ networks.

C. True or False

1. ANI is also called Weak AI.
2. AGI already exists in today's technology.
3. ASI is still a theoretical concept.
4. Reinforcement learning is similar to training animals.
5. Self-aware AI exists in present-day systems.

D. Short Answer Questions

1. Differentiate between ANI and AGI.
2. Give two real-world examples of Narrow AI.
3. What is Reactive Machine AI?
4. What is the role of Reinforcement Learning in AI?
5. Name the four AI types based on functionality.

Session 3. Domains of AI

Introduction

Artificial Intelligence (AI) is a broad field of computer science focused on building smart machines capable of performing tasks that typically require human intelligence. As AI continues to grow, it has been divided into various domains, each targeting a specific aspect of intelligence. These domains work together to enable machines to perceive, reason, learn, and act in real-world environments.

From understanding human language to recognizing images, solving problems, and even making decisions, each domain plays a vital role in the development of intelligent systems. By exploring these domains, we gain a deeper understanding of how AI mimics human behaviour and contributes to advancements in technology and society.

3.1 Introduction to AI Tools and Platforms

As Artificial Intelligence becomes more accessible and widely used, a variety of tools and platforms have emerged to support the development and deployment of AI models. These tools help simplify complex tasks like data processing, machine learning model creation, natural language processing, and more. AI platforms provide a user-friendly

environment, often with drag-and-drop features, cloud-based computing, and pre-built models, making it easier for students, developers, and professionals to work with AI.

Some popular AI platforms include Google AI, IBM Watson, Microsoft Azure AI, Amazon Web Services (AWS) AI, and OpenAI tools. These platforms offer resources for training AI systems, deploying models, and managing data effectively.

AI Tools for Various Applications

AI is being used in a wide range of real-life applications, and different tools are designed to serve specific purposes. For example:

1. Computer Vision Tools like OpenCV or TensorFlow are used in facial recognition, object detection, and image classification.
2. Natural Language Processing (NLP) Tools like spaCy or NLTK help machines understand and generate human language.
3. Voice Assistants such as Google Assistant, Siri, or Alexa use speech recognition tools for interaction.
4. Chatbots use AI platforms like Dialogflow or IBM Watson Assistant to communicate with users and answer questions.
5. Educational Tools powered by AI personalize learning experiences for students.
6. Healthcare AI Tools assist in diagnosing diseases and analysing medical images.

These tools highlight the diversity of AI applications and how specialized software is used to solve real-world problems across different fields.

3.2 Domains of AI

Major Domains of AI are as given below.

1. Data Science and Analytics – Using AI to analyse large sets of data and make predictions or decisions.
2. Natural Language Processing (NLP) – Enables machines to understand and respond to human languages.
3. Computer Vision – Allows machines to interpret and understand images and visual information.
4. Expert Systems – AI systems that mimic human decision-making using a knowledge base and set of rules.
5. Robotics – Involves AI-driven machines that can move, sense, and respond to the environment.
6. Machine Learning (ML) – A core part of AI where systems learn from data and improve over time without being explicitly programmed.

In this session we will discuss some of them.

3.3 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that enables machines to process, understand, and generate human languages. It combines linguistics, computer science, and machine learning to make computers capable of interpreting text and speech.

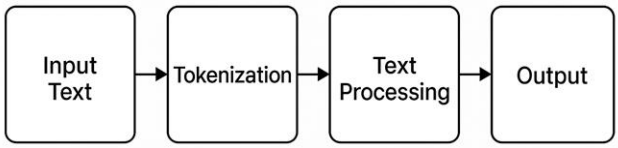


Fig. 3.1: NLP Process

Technically, NLP involves multiple tasks such as:

a) Tokenization – Breaking down text into words or sentences.
b) Part-of-Speech (POS) Tagging – Identifying nouns, verbs, adjectives, etc.
c) Named Entity Recognition (NER) – Detecting names of people, places, organizations, etc.
d) Parsing – Analysing the grammatical structure of sentences.
e) Sentiment Analysis – Detecting emotions or opinions from text.

Evolution of NLP

NLP technology has developed over a long period of time as given in following table.

Year	**Milestones**
1950s–1960s	Rule-based systems, machine translation attempts (e.g., Georgetown-IBM project).
1980s–1990s	Statistical models (e.g., Hidden Markov Models, Naïve Bayes classifiers).
2000s	Introduction of supervised learning using Support Vector Machines (SVM), decision trees.
2010s–Now	Deep learning revolution: Neural networks, Transformers, BERT, GPT models.

Modern NLP uses large pre-trained models such as:

a) BERT (Bidirectional Encoder Representations from Transformers)
b) GPT (Generative Pre-trained Transformer)
c) T5 (Text-to-Text Transfer Transformer)

Applications of NLP

Real-World Applications are as given in the following table.

Application	**Description**
Chatbots and Virtual Assistants	Use NLP for intent recognition, named entity recognition, and dialogue flow.
Machine Translation	Tools like Google Translate use Transformer models to convert languages.
Speech Recognition	Converts audio to text using NLP + speech processing (e.g., DeepSpeech).
Text Summarization	Extractive or abstractive summarization using sequence-to-sequence models.

Sentiment Analysis	Uses classifiers like SVM, LSTM, or BERT for opinion detection.
Autocorrect and Prediction	Uses N-gram models or neural networks for predictive text.

NLP and Its Relationship with AI

NLP is a core domain of Artificial Intelligence that enables intelligent systems to communicate using language. It connects other AI subfields such as:

a) Machine Learning – Used to train models on large language datasets.

b) Cognitive Computing – Imitates human thought processes using NLP.

c) Robotics and Voice Interfaces – Enable voice-driven interaction using NLP.

Through NLP, AI systems can analyse unstructured data, which is most of the data humans produce.

Examples of NLP

Several real-life commercial applications make use of NLP. Some of them are as mentioned below.

Example	**NLP Task Involved**	**Underlying Technology**
ChatGPT	Text generation, context understanding	Transformer, Reinforcement Learning
Gmail Smart Compose	Next-word prediction	Neural language models (e.g., LSTM)
Google Assistant	Intent recognition, speech-to-text	NLP + Speech APIs
Google Translate	Language translation	Sequence-to-sequence Transformer models
Grammarly	Grammar correction, style suggestions	Rule-based + ML classifiers
Twitter Sentiment Tool	Sentiment classification	BERT or LSTM models

Case Study: NLP in Customer Support – Chatbot for E-commerce

Case Title:

Improving Customer Service Using an AI-Powered Chatbot with NLP

Background:

An e-commerce company was struggling to keep up with thousands of customer support queries.

Problem:

1. Slow responses
2. Low customer satisfaction
3. Overworked support team

Solution Using NLP:

The company implemented an AI-powered chatbot using Natural Language Processing (NLP) to automatically understand and reply to customer queries.

NLP Features Used:

Feature	Role in Chatbot
Intent Recognition	Identify the customer's goal (e.g., order tracking).
Entity Extraction	Extract key information like product names or order numbers.
Context Management	Maintain flow of conversation.
Text Generation	Create natural, readable replies.

Python Code Example

Below is a simple Python example using Hugging Face Transformers to simulate a chatbot that understands and responds to customer queries:

Requirements:

Make sure to install the libraries first:

pip install transformers torch

Python Code:

```
from transformers import pipeline

# Load a pre-trained conversational model
chatbot = pipeline ("conversational", model="microsoft/DialoGPT-medium")

from transformers import ConversationalPipeline, Conversation

# Simulate a conversation
user_input = "Where is my order?"
conversation = Conversation(user_input)

# Get chatbot response
response = chatbot(conversation)
print("Bot:", response.generated_responses[-1])
```

Result of Deployment:

Metric	Before	After
Avg. Response Time	5–10 minutes	Instant
Auto-handled Queries	0%	70%
Customer Satisfaction Score	75%	90%
Agent Workload	High	Reduced by 60%

Using a Python-based NLP chatbot integrated with an AI platform like Hugging Face allowed the company to automate and personalize customer support. This improved speed, reduced costs, and greatly enhanced customer satisfaction.

3.4 Computer Vision

Computer Vision (CV) is a field of Artificial Intelligence that enables machines to interpret and understand the visual world—such as images, videos, and real-time scenes—just like humans do. It involves developing algorithms that allow computers to:

1. Recognize objects
2. Understand scenes
3. Detect patterns
4. Analyse movement

Technically, CV tasks include:

1. Image classification – Identifying what is in an image
2. Object detection – Locating objects in an image
3. Segmentation – Separating objects from the background
4. Facial recognition – Identifying or verifying faces

Evolution of Computer Vision

Year	Milestones
1960s–1980s	Basic image processing, pattern recognition using pixel-based techniques
1990s	Edge detection, feature extraction (e.g., SIFT, HOG), image databases
2000s	Machine learning-based image classification using SVM and KNN
2012–Present	Deep Learning revolution with CNNs (Convolutional Neural Networks), achieving near-human accuracy in vision tasks

Applications of Computer Vision

Real-World Use Cases are as given below:

Application	Description	CV Technique/Model Used
Face Unlock (Phones)	Verifies identity using facial landmarks and embedding	FaceNet, MTCNN
Medical Imaging	Detects tumours, fractures, and diseases in X-rays and MRIs	CNNs, U-Net (for segmentation)
Autonomous Vehicles	Detects lanes, pedestrians, traffic signs in real-time	YOLO, OpenCV, Deep Learning
Security Systems	Motion detection, person recognition, license plate recognition	Haar cascades, object tracking
Retail Analytics	Analyses footfall, product interaction in stores	Object detection, pose estimation

Computer Vision and its relationship with AI

Computer Vision is a core domain of Artificial Intelligence focused on perceptual tasks. It combines AI + image processing + machine learning + deep learning to mimic how humans perceive and understand images.

It often integrates with:

1. Natural Language Processing for visual question answering.
2. Robotics for visual navigation.
3. Reinforcement learning for visual decision-making tasks.

AI provides the intelligence layer, while CV provides the "eyes" of the system.

Examples of Computer Vision

Some commercial examples of CV are as mentioned below.

Example	Use Case	Underlying Technology
Google Photos	Image recognition and face grouping	CNN-based image embedding
Tesla Autopilot	Lane and object detection in real time	Real-time object detection (YOLO)
Facebook Tagging	Automatic face detection and tagging	Face recognition using deep nets
Instagram Filters	Face tracking for AR filters	Facial landmark detection
CCTV Systems	Person tracking and intrusion detection	Motion analysis, object tracking

Case Study: Computer Vision in Retail – People Counting in a Store

Case Title:

Using Computer Vision for Customer Footfall Analysis in Retail

Background:

A retail chain wants to understand customer behaviour by analysing how many people enter the store and how long they stay. This data helps with store layout, staff planning, and marketing.

Problem:

I. Manual tracking is time-consuming and error-prone
II. No real-time insights into customer traffic

Solution Using Computer Vision (CV):

Install cameras at store entrances and use a CV-based people detection and counting system using OpenCV and a pre-trained object detection model.

CV Features Used:

Feature	Role
Object Detection	Identify people in the camera feed
Bounding Boxes	Mark individuals visually using rectangles
Object Tracking	Track movements to avoid double counting
Real-Time Processing	Analyze live video streams for instant footfall updates

Requirements:

```
pip install opencv-python
```

Also, you need:

1. yolov3.weights and yolov3.cfg from the official YOLO website.
2. The COCO dataset labels where class ID 0 represents "person".

Python Code Example: People Detection Using OpenCV and YOLOv3

```
import cv2
# Load YOLO pre-trained weights and config
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# Load camera/video
cap = cv2.VideoCapture(0)  # 0 = default webcam

while True:
    ret, frame = cap.read()
    height, width = frame.shape[:2]

    # Convert frame to blob for YOLO
    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)
    net.setInput(blob)
    outputs = net.forward(output_layers)

    # Process outputs
    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = scores.argmax()
            confidence = scores[class_id]

            # Class ID 0 = 'person' in COCO dataset
            if class_id == 0 and confidence > 0.5:
                center_x, center_y, w, h = (detection[0:4] * [width, height, width, height]).astype("int")
      x = int(center_x - w / 2)
      y = int(center_y - h / 2)
      cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cv2.imshow("People Detection", frame)
    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Results:

Metric	Before CV	After CV
Customer Count Accuracy	65% (manual)	95% (automated)
Real-Time Insights	Not available	Live dashboard
Staff Scheduling Optimization	Manual	Data-driven

This case shows how Computer Vision can be used for real-time decision-making in retail by automating people counting using a camera and an AI model. It reduces labour, increases accuracy, and leads to smarter business decisions.

1.5 Robotics

Robotics is a multidisciplinary field that combines engineering, computer science, and artificial intelligence (AI) to design, build, and program machines—called robots—that can perform tasks autonomously or semi-autonomously.

A robot typically includes:

1. Sensors to perceive the environment.
2. Actuators motors and parts that allow movement.
3. Controller processors execute programs.
4. Software/AI to make decisions or perform tasks.

Robotics allows machines to mimic human actions, senses, and reasoning.

Evolution of Robotics

Year	Milestones
Ancient Times	Mechanical devices like water clocks and automata in Greece and China
1950s–1970s	First industrial robots (e.g., Unimate in 1961 for car manufacturing)
1980s–1990s	Introduction of mobile robots and embedded control systems
2000s–2010s	Use of microcontrollers, sensors, and early AI for more complex robotics
2010–Present	Integration of AI, CV, NLP, and ML into humanoid robots, drones, and autonomous vehicles

Applications of Robotics

Commercial fields using Robotics and related technologies are given below:

Application	Description	Related Technology
Manufacturing	Welding, painting, assembling in factories	Industrial robots, PLCs
Healthcare	Surgery assistance, elderly care, medicine delivery	Surgical robots (e.g., Da Vinci)
Agriculture	Autonomous plowing, seeding, crop monitoring	Drones, sensor-based robots
Military & Defense	Reconnaissance, bomb disposal, surveillance	UGVs, UAVs, AI targeting systems

Household	Cleaning, vacuuming, home security	Service robots (e.g., Roomba)
Education & Research	STEM learning kits, robotic labs	LEGO Mindstorms, Raspberry Pi bots

Robotics relation to AI

Robotics and AI are closely connected. While robotics focuses on the body, AI focuses on the brain. AI empowers robots to:

1. Make decisions using ML algorithms.
2. Perceive environments using Computer Vision and NLP.
3. Adapt to change using Reinforcement Learning.
4. Interact intelligently using speech recognition and planning.

Robotics = Mechanical + Electrical Systems + AI Algorithms

Examples of Robotics in Real Life

Robot Name	Use Case	AI Feature
Boston Dynamics Spot	Surveillance, terrain navigation	Vision, balance, path planning
Roomba (iRobot)	Smart vacuum cleaning	Room mapping, obstacle avoidance
Sofia the Robot	Human-like conversation	NLP, facial expression recognition
Da Vinci Surgical Robot	Precision surgeries	Motion control with feedback systems
Amazon Robotics	Warehouse item sorting	Navigation + machine learning
Mars Rovers	Planet exploration	Autonomous movement, remote AI control
Mitra Robot (Invento Robotics)	Healthcare & Hospitality	Facial Recognition, NLP
Daksh (DRDO)	Bomb disposal and hazardous object handling	Remote control, navigation, manipulation arms
Manav – India's first 3D-printed humanoid robot	Used for human-robot interaction studies	Voice recognition, camera sensors, AI scripts

Case Study: Service Robot in Hospitals – "Mitra" by Invento Robotics

Case Title:

Improving Patient Interaction and Safety Using a Humanoid Robot

Background:

During the COVID-19 pandemic, hospitals in India needed to reduce human-to-human contact. The Mitra robot was deployed in several hospitals (e.g., Yatharth Hospital, Noida) to help doctors interact with patients safely.

Problem:

1. Direct interaction between patients and healthcare staff was risky.
2. Hospitals lacked automation for basic tasks like greeting or data collection.

Solution Using Robotics:

Mitra, a human-sized interactive robot, was used to:

1. Greet patients using facial recognition and speech.
2. Collect data using touchscreens and voice commands.
3. Enable remote doctor-patient communication through video calls.

Technologies Used in Mitra:

Function	Technology
Face Recognition	Computer Vision + Deep Learning (CNNs)
Voice Interaction	NLP, Text-to-Speech and Speech Recognition
Navigation	Obstacle Avoidance using IR and Lidar Sensors
Touchscreen Interface	Android-based application
Communication	Wi-Fi + Cloud integration for video calling

Python Code Example: Simulated Greeting Robot

Here's a simple Python simulation of a service robot that greets users using basic voice and face detection:

Requirements:

```
pip install opencv-python pyttsx3
```

Python Code:

```
import cv2
import pyttsx3
# Initialize text-to-speech engine
engine = pyttsx3.init()

# Function to greet
def greet(name="Visitor"):
    message = f"Hello {name}, welcome to the hospital. Please wear
a mask."
    print("Robot:", message)
    engine.say(message)
    engine.runAndWait()

# Start camera for face detection
face_cascade    =    cv2.CascadeClassifier(cv2.data.haarcascades    +
'haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)

print("Starting robot camera...")
```

```
while True:
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:
        # Draw rectangle around face
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        greet()
        break  # Greet once per session

    cv2.imshow("Hospital Robot View", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Results of Deployment

Metric	Before	After
Doctor-patient contact risk	High	Reduced by 70%
Patient greeting time	Manual, inconsistent	Automated, consistent
Data collection	Manual	Voice and touchscreen-based

This case illustrates how AI-powered robots like Mitra are used in real-world healthcare in India. By combining face recognition, speech interaction, and mobility, robotics helps reduce health risks while maintaining quality care.

1.6 Expert Systems

An Expert System is a computer program designed to simulate the decision-making abilities of a human expert. It uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise. Figure 3.2 shows a block diagram of an expert system.

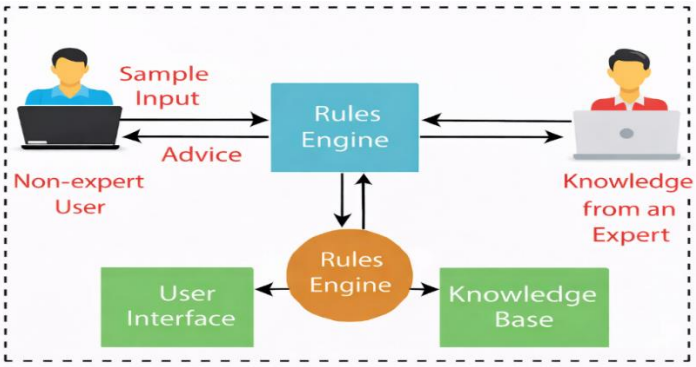


Fig. 3.2: Block Diagram of Expert System

Key Components:

1. Knowledge Base: Contains domain-specific facts and rules.
2. Inference Engine: Applies logical rules to the knowledge base to deduce new information or solve problems.
3. User Interface: Allows interaction between the user and the system.
4. Explanation Facility (optional): Explains the reasoning process of the system.

Evolution of Expert Systems

Year	Features
1960-70s	Research on Artificial Intelligence led to the first Expert Systems like DENDRAL (chemistry) and MYCIN (medical diagnosis).
1970-80s	Widespread industrial and commercial use. Tools like XCON by Digital Equipment Corporation streamlined product configurations
1980-90s	Integration with databases, GUI, and the internet. Decline in popularity due to the rise of Machine Learning and Data-Driven AI.
1990-Present	Expert Systems are used in hybrid systems combining rule-based reasoning with AI/ML

Applications of Expert Systems

Expert Systems are widely used across multiple domains:

Field	Application
Medicine	Diagnosis and treatment recommendation (e.g., MYCIN)
Engineering	Fault diagnosis in machinery
Finance	Credit scoring, fraud detection
Agriculture	Pest control, crop planning
Customer Service	Automated support systems
Education	Intelligent tutoring systems

Relationship between Expert Systems and AI

Expert Systems are a subfield of Artificial Intelligence (AI). They are one of the earliest successful applications of AI. While modern AI heavily relies on machine learning and data, Expert Systems use symbolic AI that is, rules and logic.

AI vs. Expert Systems:

Feature	Expert Systems	General AI
Knowledge Source	Human expert (manual)	Data-driven
Learning Capability	Typically, static	Can learn and improve
Explainability	High	Often limited
Inference Style	Rule-based	Rule-based or statistical

Examples of Expert Systems

System	Domain	Function

MYCIN	Medical	Diagnose bacterial infections
DENDRAL	Chemistry	Analyze mass spectrometry data
XCON	Computer Configuration	Assist in configuring VAX computers
Kisan Knowledge Management System (KKMS)	Agriculture	Helps farmers get crop-specific advice, pest control tips, and weather info. Developed by ICAR (Indian Council of Agricultural Research).
eSagu	Agriculture	An IT-based personalized agro-advisory system providing scientific farming advice. Piloted in Andhra Pradesh.
VEDAS (Visual Exploitation of Data and Archival System)	Remote Sensing / Space Research	Developed by ISRO for analyzing satellite data, mineral exploration, and disaster management.
GISTNIC (General Information Service Terminal for National Informatics Centre)	Government Services	Provides support to government officials for policy decisions using rule-based inference.
AIIMS Expert System	Healthcare	Assists doctors with diagnostic support and treatment suggestions at AIIMS hospitals.

Case Study: Expert System for Crop Recommendation – "AgroExpert"

Problem Statement:

Farmers in India often face difficulty choosing the right crop for the season based on soil, rainfall, and temperature. An expert system can help make this decision, improving yield and efficiency.

Objective:

To develop a simple rule-based expert system that recommends crops based on:

1. Soil type
2. Rainfall level
3. Temperature range

Rule Base (Sample Rules):

Soil Type	Rainfall	Temperature	Recommended Crop
Loamy	High	Moderate	Rice
Sandy	Low	High	Millets
Clay	Medium	Low	Wheat
Black	High	High	Cotton

Python Code Example:

Simulating a basic rule-based expert system.

```
def recommend_crop(soil, rainfall, temperature):
```

```
    rules = [
        {"soil": "loamy", "rainfall": "high", "temperature":
"moderate", "crop": "Rice"},
        {"soil": "sandy", "rainfall": "low", "temperature":
"high", "crop": "Millets"},
        {"soil": "clay", "rainfall": "medium", "temperature":
"low", "crop": "Wheat"},
        {"soil": "black", "rainfall": "high", "temperature":
"high", "crop": "Cotton"},
    ]

    for rule in rules:
        if (rule["soil"] == soil.lower() and
            rule["rainfall"] == rainfall.lower() and
            rule["temperature"] == temperature.lower()):
            return f"Recommended Crop: {rule['crop']}"

    return "No recommendation found for the given conditions."

# Example usage
soil_type = input("Enter soil type (Loamy, Sandy, Clay, Black):
")
rainfall = input("Enter rainfall level (Low, Medium, High): ")
temperature = input("Enter temperature level (Low, Moderate,
High): ")

result = recommend_crop(soil_type, rainfall, temperature)
print(result)
```

Output:

```
Enter soil type (Loamy, Sandy, Clay, Black): Sandy
Enter rainfall level (Low, Medium, High): Low
Enter temperature level (Low, Moderate, High): High
Recommended Crop: Millets
```

This system is easy to use by farmers via mobile apps in local languages. It reduces dependency on physical experts in remote areas. It promotes scientific agriculture.

1.7 Examples of AI Platforms by Application

Healthcare

Platform	Use Case	Example
IBM Watson Health	Diagnosis support, cancer treatment plans	Used by hospitals for oncology insights
Google DeepMind	Disease prediction, eye disease analysis	Used by NHS (UK) for early diagnosis

Aidoc	Radiology AI	Detects medical conditions in CT scans

Agriculture

Platform	Use Case	Example
Plantix	Crop disease identification	Mobile app used by Indian farmers
Microsoft Azure FarmBeats	Precision farming	IoT + AI platform for improving crop yield
IBM Watson Decision Platform for Agriculture	Weather insights, pest detection	Used in agri-tech solutions in India

Education

Platform	Use Case	Example
Squirrel AI	Adaptive learning	Used in China for K-12 education
Google Cloud AI	Personalized learning, analytics	Used by EdTech companies
Amazon SageMaker	Build education-focused ML models	Predicting student performance
ConveGenius	AI-driven EdTech for low-income schools	Used in UP government schools

Finance

Platform	Use Case	Example
DataRobot	Credit scoring, fraud detection	Used by banks to automate credit risk
SAS Viya	Risk analytics, forecasting	Popular in large banks and insurance firms
Upstart AI	Loan approval based on AI scoring	Used by Indian NBFCs and startups
Razorpay Rize	Fraud detection and payment automation	Protects payment gateways in India

Retail & Ecommerce

Platform	Use Case	Example
Google Vertex AI	Recommendation engines, demand forecast	Used by Flipkart for product suggestions
Amazon Personalize	Product and content recommendations	Used in e-commerce personalization
Salesforce Einstein	Customer behavior analysis	Used in marketing automation

Autonomous Systems & Robotics

Platform	Use Case	Example
NVIDIA Isaac	Robot simulation,	Used in warehouse automation

	autonomous control	
ROS (Robot Operating System)	Open-source robotics framework	Used in drones and self-driving research
OpenAI Gym	Reinforcement learning environments	Used in autonomous vehicle simulations

Voice Assistants and NLP

Platform	Use Case	Example
Google Dialog Flow	Chatbots and virtual agents	Used in Indian Railways and banks
Microsoft LUIS	Language understanding	Integrated into customer support systems
Amazon Lex	Conversational interfaces	Used in Alexa and customer support bots

Government / Public Sector

Platform / Program	Use Case	Example
Bhuvan (ISRO)	AI + GIS for land and disaster management	Used in Smart City planning and agriculture
RAISE 2020, IndiaAI Initiative	National AI research and deployment platform	Developed by MeitY, NITI Aayog
NIC Chatbot (Rashtriya eSampark)	Public grievance redressal via chatbot	Integrated in government portals
AI in FASTag (NHAI)	Toll monitoring, fraud detection	AI detects wrong vehicle classification

3.8 Examples of Programming Languages Used to Build AI Applications

AI applications are built using programming languages that support numerical computing, machine learning libraries, natural language processing, and neural network development. Different languages serve different aspects of AI, such as prototyping, performance, and production deployment.

Programming Language	Features	Use Cases
Python Programming	• Easy syntax and readability • Rich ecosystem of AI/ML libraries: TensorFlow, PyTorch, Scikit-learn, NLTK, OpenCV • Massive community and educational resources • Integration with data tools such as pandas and NumPy.	• Machine Learning models • Natural Language Processing • Computer Vision • Deep Learning
R Programming	• Designed for statistical computing and data analysis • Rich set of visualization tools and	• Statistical modelling • Predictive analytics • Academic AI

	libraries (e.g., ggplot2) • Packages like caret, randomForest, and nnet for AI tasks	research
Java Programming	• Strong in large-scale enterprise applications. • Libraries like Weka, Deeplearning4j, MOA. • Platform independence and strong memory management.	• Production-grade AI systems. • AI in enterprise software. • Mobile AI via Android is possible.
C++ Programming	• High-performance language with low-level memory control • Used in AI engines where speed is critical such as, gaming, robotics. • Backend development of some AI frameworks such as TensorFlow.	• Real-time systems • Autonomous vehicles • Game AI
JavaScript Programming	• Strong in large-scale enterprise applications. • Libraries like Weka, Deeplearning4j, MOA. • Platform independence and strong memory management.	• Production-grade AI systems. • AI in enterprise software. • Mobile AI via Android is possible.
Julia Programming	• Designed for high-performance numerical analysis • Combines the speed of C with Python-like syntax • Libraries like Flux.jl and MLJ.jl for ML	• Scientific computing • High-speed AI prototyping • Financial modeling

Summary Table

Language	**Strength**	**Key Libraries/ Frameworks**	**Common Use Cases**
Python	Easy, powerful libraries	TensorFlow, PyTorch, Scikit-learn	ML, NLP, CV, Deep Learning
R	Statistical computing	caret, ggplot2, randomForest	Data analysis, Predictive models
Java	Scalable enterprise solutions	Weka, Deeplearning4j	Enterprise AI, mobile AI
C++	Performance-critical tasks	Used in TensorFlow backend	Robotics, Gaming AI
JavaScript	Web and client-side AI	TensorFlow.js, Brain.js	AI in browsers, chatbots
Julia	High-speed numerical computing	Flux.jl, MLJ.jl	Research, finance, simulations

Summary

- AI domains: Data Science, NLP, Computer Vision, Expert Systems, Robotics, ML.
- NLP: Understanding & generating human language (chatbots, translation, sentiment analysis). Evolution: rule-based → statistical → deep learning (BERT, GPT).
- Computer Vision: Enables machines to "see" (face unlock, medical imaging, autonomous vehicles). Evolution: image processing → CNN-based systems.
- Robotics: Combines hardware + AI for automation (industrial robots, medical robots, exploration robots). Example: Mitra Robot in Indian hospitals.
- Expert Systems: Rule-based AI mimicking experts (MYCIN, DENDRAL, AgroExpert). Still used in agriculture, medicine, and government services.
- AI Platforms by Sector: Healthcare: IBM Watson Health, DeepMind, Agriculture: Plantix, Azure FarmBeats, Education: Squirrel AI, Google Cloud AI, Finance: DataRobot, Upstart, Retail: Vertex AI, Amazon Personalize, Robotics: NVIDIA Isaac, ROS, Government: ISRO Bhuvan, NIC Chatbots.
- Programming Languages for AI: Python (ML, NLP, CV), R (statistics, predictive models), Java (enterprise AI), C++ (performance, robotics), JavaScript (web AI), Julia (scientific computing).

Check Your Progress

A. Multiple Choice Questions

1. Which AI domain is used in Google Translate? (a) Robotics (b) NLP (c) Expert Systems (d) Computer Vision
2. Which of the following is a Computer Vision application? (a) Siri (b) Grammarly (c) Face Unlock (d) ChatGPT
3. Which AI domain mimics expert decision-making? (a) Robotics (b) Computer Vision (c) Expert Systems (d) Machine Learning
4. Which language is most widely used for AI applications? (a) C (b) JavaScript (c) Python (d) PHP
5. Which robot was used in Indian hospitals during COVID-19? (a) Manav (b) Mitra (c) Sophia (d) Spot

B. Fill in the blanks

1. NLP stands for _____.
2. _____ enables machines to "see" and interpret images.
3. An _____ System uses a knowledge base and inference engine.
4. _____ combines hardware and AI for automation.
5. Python libraries like TensorFlow and _____ are widely used for AI.

C. True or False

1. Expert systems are rule-based AI systems.
2. Computer Vision is only used in healthcare.
3. NLP allows machines to understand human language.
4. Robotics deals only with software, not hardware.
5. R language is commonly used for statistical AI tasks.

D. Short Answer

1. Define Natural Language Processing with an example.
2. Mention two applications of Computer Vision.
3. What are Expert Systems? Give one example.
4. Write two uses of AI in Robotics.
5. List any three programming languages used in AI.

Module 2. Python Programming

Module Overview

Python is a very simple programming language, developed by Guido van Rossum in 1989. It is named after the comedy television show Monty Python's Flying Circus and not after the Python snake. It is easy to learn for all who are new to programming.

Python is a dynamic, interpreted (bytecode-compiled) language. It is platform independent, free and open source, easy to learn, achieve more tasks through less coding, and fast in execution. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible.

Python is a general-purpose language, which can be used in various domains including: Web applications, Big data applications, Data science, Desktop software, Mobile apps, Machine learning and AI. The SQL can be used for querying data from relational databases. Python is a multifunctional programming language which is object oriented, functional programming and also declarative programming language.

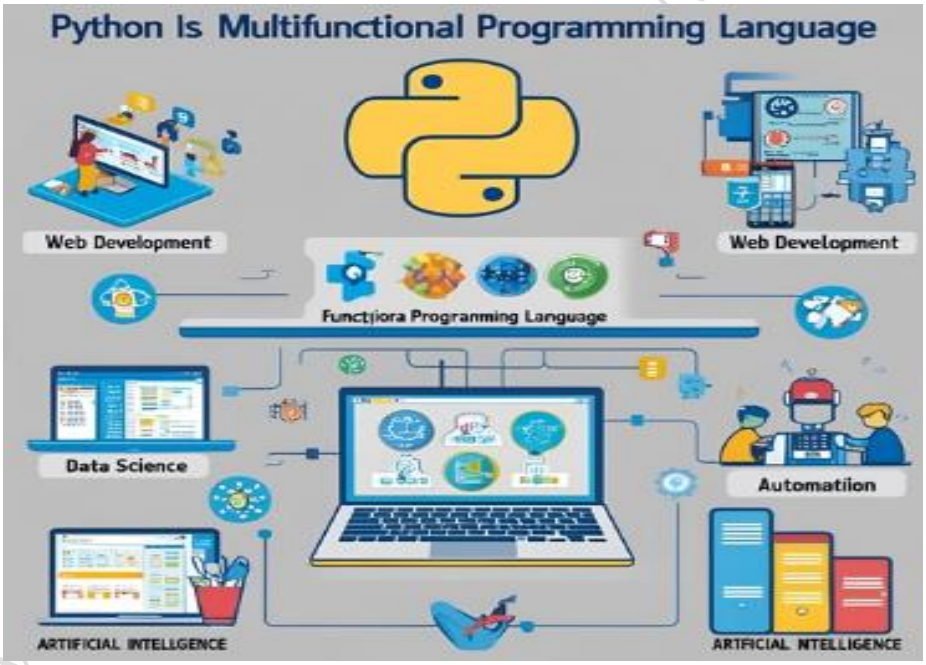


Fig. 1.0: Python: A Multifunctional Programming Language

Learning Outcomes

After completing this module, you will be able to:

- Understand Python fundamentals such as features of Python, syntax, keywords, variables, and writing simple programs.
- Use control flow statements like if-else, for loop, while loop, and conditional expressions to control program execution.
- Identify and use different data types such as integers, floats, strings, lists, tuples, and dictionaries in Python programs.

- Define and use functions to organize code, pass arguments, and return values for modular programming.
- Develop simple Python programs using basic concepts learned in the module.

Module Structure

Session 1. Python Fundamentals

Session 2. Control Flow in Python

Session 3. Python Data Types

Session 4. Python Functions

Session 1. Python Fundamentals

1.1 INTRODUCTION

A programming language is a set of grammatical rules for instructing a computer to perform specific tasks. It is used to code the programs in the computer to perform a specific task to produce the desired output. There are many different programming languages such as BASIC, Pascal, C, C++, Java, Ruby, Python, and many others. The other programming languages are designed for specific purposes, while Python is a flexible and multipurpose programming language.

In this session you will understand the basics of Python programming, its features, and installation of Python. You will also learn the Keywords, Identifiers, and Variables in Python. The Data Types, Operators, and Statements in Python is also explained. You will be able to run the sample code in the Python interpreter as well as write the programs in IDLE and run to get the output.

1.2 PYTHON PROGRAMMING

A computer program is a set of instructions written in any programming language to perform a specific task after its execution by a computer.

The first version of Python was labeled as "0.9.0". The first official version 1.0 was released in January 1994. The current version of Python is 3.13, released on October 7, 2024.

1.2.1 Python for AI

Artificial intelligence (AI) is the most recent technology. Many AI applications are being developed in various area. There are various programming languages like Lisp, Prolog, C++, Java and Python, which can be used for developing applications of AI. It is the robust programming language used in many domains from web applications, data analysis, data science, machine learning, and AI. Python is the most suitable language because of the following features.

1.2.2 Features of Python programming

Python is an interpreted, high-level, general-purpose programming language. It allows solving complex problems in less time and fewer lines of code. It is simple and easy-to-learn. Python can be used to build all types of applications ranging from small and

simple scripts to complex machine learning algorithms. Python source code is available under the GNU General Public License (GPL).

Readable and Easy to Learn – Python is a very readable language. Python programs are easy to understand as they have a clearly defined syntax and relatively simple structure. It is an interpreted language, as Python programs are executed by an interpreter.

Interactive Mode – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Case Sensitive – Python is a case-sensitive language, means, the variables "NUMBER" and "number" are different in Python. Python uses indentation for blocks and nested blocks.

Free and Open Source – Python is a free and open source programming language. It can be downloaded from its official website for free to use.

Cross platform – Python is portable and platform independent. It can run on various operating systems such as Mac, Windows, Linux, Unix and any hardware platforms. This makes it a cross platform and portable language.

Large standard library – Python comes with a large standard library of predefined functions that has some handy codes and functions which can be used while writing code in Python. Python is also helpful in web development. Many popular web services and applications are built using Python.

Supports exception handling – Python supports exception handling which means we can write less error prone code and can test various scenarios that can cause an exception later on.

Portability and Compatibility – Python can run on a wide variety of operating systems and hardware platforms, has the same interface on all platforms.

Extendable. We can add low-level modules to the Python interpreter. These modules enable programmers to customize their tools to be more efficient.

Automatic memory management – Python supports automatic memory management which means the memory is cleared and freed automatically.

1.2.3 Applications of Python

Python is used for a large number of applications as below:

1.2.4 Working with Python

Python is a cross-platform programming language, means, it runs on multiple platforms like Windows, MacOS, Linux and has even been ported to the Java and .NET virtual machines. It is also available online in a cloud environment.

To write and run a Python program, we need to have a Python interpreter installed on our computer.

Practical Activity 1.1. Install the Python compiler (binaries)

Resources required

A Desktop or Laptop computer with operating system installed, Internet Connectivity

Procedure

Step 1. Download Python from its official website *https://www.python.org/*downloads. Select appropriate download link as per Operating System [Windows 32 Bit/64 Bit, Apple iOS].

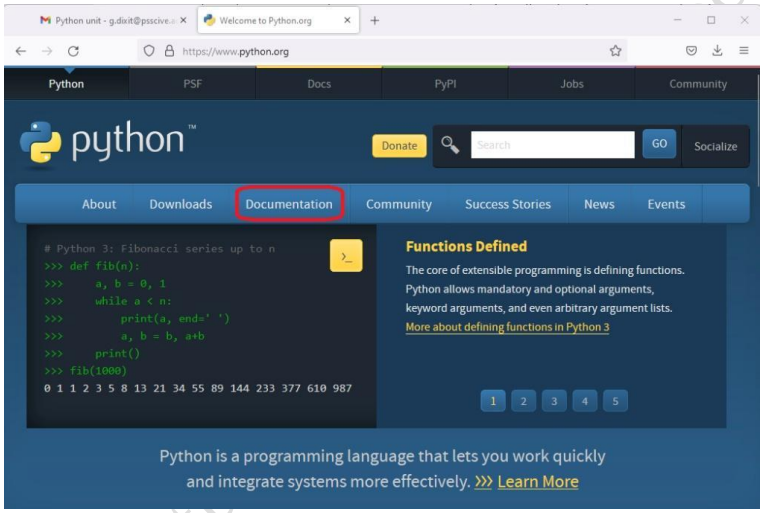


Fig. 1.1 Official website of Python

Step 2. For Windows 64 Bit OS, download latest version using the link

https://www.python.org/ftp/python/3.13.2/python-3.13.2-amd64.exe

Step 3. Download the installer. Once it is downloaded, run the Python installer.

Step 4. Check the **Install launcher for all users** check box. Further, you may check the **Add Python 3.13.2** to path check box to include the interpreter in the execution path.

Step 5. Select **Customize installation**.

Step 6. After selecting the Advanced options, click **Install** to start installation. Follow the instructions to complete the installation process.

Step 7. Once the installation is over, a window will appear showing the **Python Setup Successful**.

Run in the Integrated Development Environment (IDE)

After installing Python, you can start developing Python programs. An alternate way to reach python is to search for *"Python"* in the start menu and clicking on IDLE (Python 3.13.3 64-bit). You can start coding in Python using the Integrated Development and Learning Environment (IDLE**)** as shown in Figure 1.4.

IDLE is Python's Integrated Development and Learning Environment. It is coded in Python itself, using the tkinter GUI toolkit. It is cross-platform i.e. it works mostly same on Windows, Linux, and MacOS. Python shell window (interactive interpreter) with colorizing of code input, output, and error messages.

1.3 Python Interpreter Mode

Python interpreter can be used in two modes – ***Interactive mode*** and ***Script mode***. Interactive mode allows execution of individual statements instantaneously. Whereas, the script mode allows writing more than one instruction in a file called Python source code file that can be executed.

1.3.1 Interactive Mode

To work in the interactive mode, simply type the Python statement on the **>>>** prompt directly and press ***Enter*** key. The interpreter executes the statement and displays the result(s), as shown in Figure 1.4. Working in the interactive mode is convenient for testing a single line code for instant execution. But in the interactive mode, we cannot save the programming code for future use and we have to retype the statements to run them again.

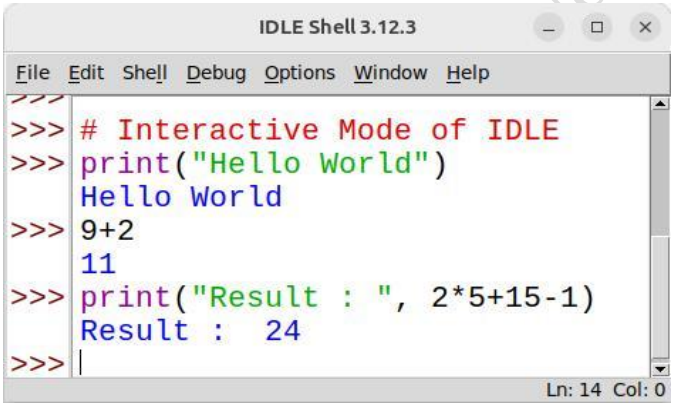


Fig. 1.2: Interactive mode of IDLE

You can see the above example, Python IDLE Shell account has **>>>** as Python prompt, where simple mathematical expressions and single line Python commands can be written and can be executed simply by pressing enter.

The first statement print ("Hello World") written on the first Python prompt shows Hello World as output in the next line.

The second expression 9+2 written on the second Python prompt shows 11 as output in the next line.

The third statement print("Result:", 2*5+15-1) written on the fourth Python prompt shows

Result: 24

Assignment

Find the result of (75+85+65)/3 i.e. the average of three marks

Find the result of 22/7 * 5 * 5 i.e. the area of circle having radius as 5

Find the result of "RAVI"+"Kant"

Find the result of "###" * 3

1.3.2 Script Mode

In the script mode, you have to write a Python program in the editor and save the file with ***.py*** extension. Then use the interpreter to execute it by entering the file name on the python prompt (>>>). By default, the Python program is saved in the Python installation folder. Let us see steps to run Python interpreter in script mode.

Working in interactive mode is easy to test small pieces of code immediately. But for coding large programs it is always better to use the script mode, so that you can save and modify the code whenever required.

Note: Result produced by Interpreter in both the modes, viz., Interactive and script mode is exactly the same.

Python Script/Program: Python statements written in a particular sequence to solve a problem is known as Python Script/Program.

To write a Python script/program, open a new file using **File >> New File**, type a sequence of Python statements for solving a problem, save it with a meaningful name using **File >> Save**, and finally run the program using **Run >> Run Module** to view the output of the program.

Following steps illustrate how to write your first python program.

Step 1. Create a new file by clicking on **File > New File** menu in IDLE as shown in Figure 1.3.

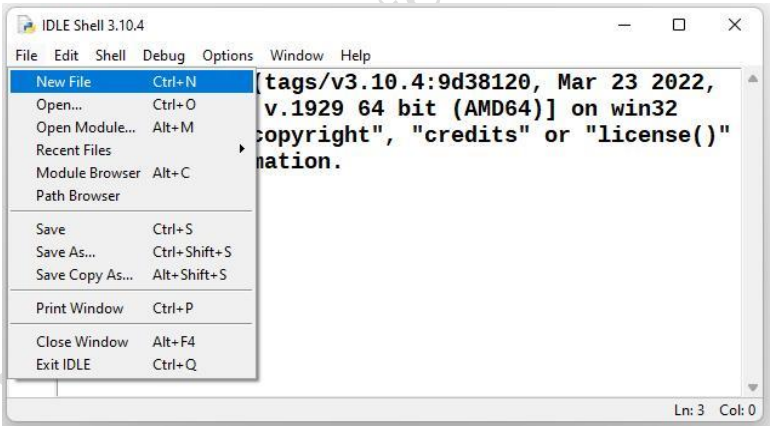


Fig. 1.3: Creating a new file in Python

Step 2. A new file will open in editor. Enter the Python code as shown in Figure 1.4.

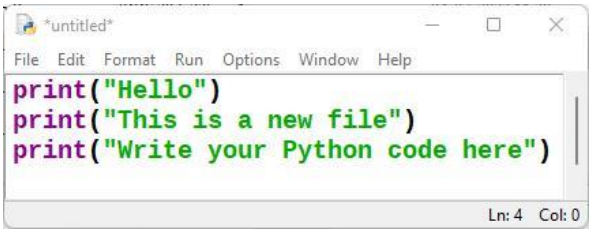


Fig. 1.4: Coding a Python program

Step 3. To save the file, click on **File > Save** menu option. The **Save as** window will open as shown in Figure 1.5. Select the folder in which you want to save the file. Type the file name and click on Save button as shown in Figure 1.5.

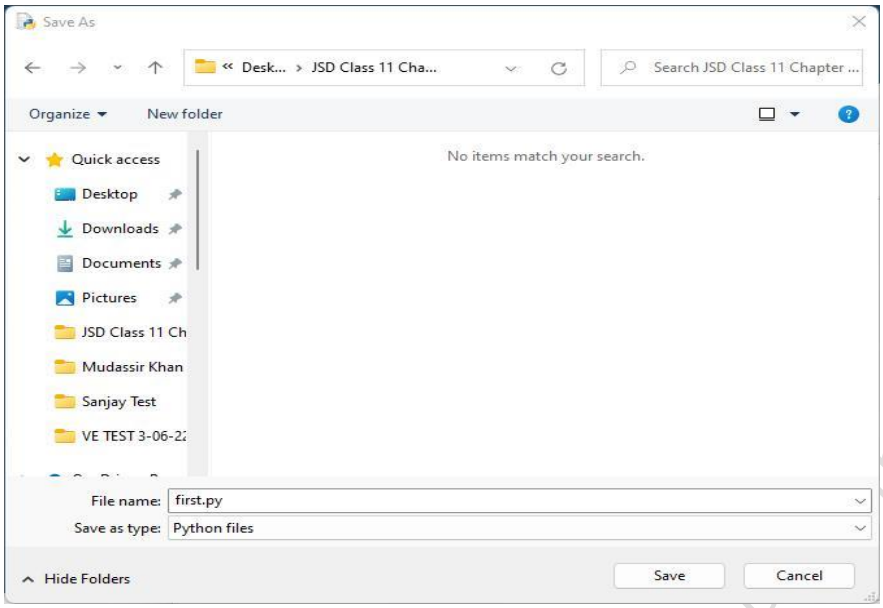


Fig. 1.5: Saving a Python program

Step 4. After saving the file, click **Run Module** from the Run menu as shown in Figure 1.6. You can also press **F5 key** to run the program.

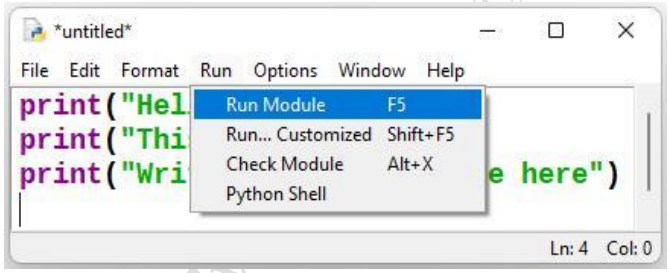


Fig. 1.6: Running a Python program

Step 5. The output appears on shell as shown in Figure 1.7.

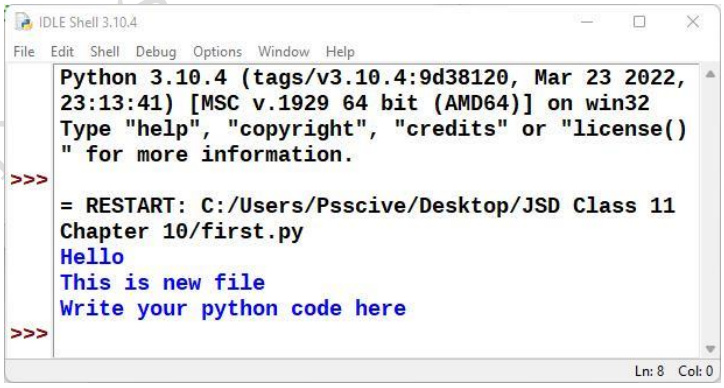


Fig. 1.7: Showing output of Python program

ou are familiar with both modes of Python Interpreter, let us see the general structure of the Python Program.

1.4 Structure of a Python program

In general, the interpreter reads and executes the Python statements line by line i.e. sequentially, however there are some statements that can alter this behaviour like conditional statements.

Python Statement and Comments

Python programs consist of statements and comments. The indentation is important in statements. The comments are used to explain the specific code.

Python Statement

Instructions written in the source code for execution are called statements. Python statements are written line by line such that one statement ends up in that line only. The interpreter considers the *'new line character'* as the terminator of one instruction. However, writing multiple statements per line is also possible in script mode of Python IDLE.

There are different types of statements in the Python programming language like Assignment statement, Conditional statement, Looping statements etc. These help the user to get the required output. For example, n = 50 is an assignment statement.

Multi-line statement

In Python, the end of a statement is marked by a newline character.

However, Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;), continuation character slash (\). When we need to do long calculations and cannot fit these statements into one line, we can make use of these characters.

Table 1.1

Type of Multi-line Statement	**Usage**
Using Continuation Character (/)	S = 1 + 2 + 3 + \ 4 + 5 + 6 + \ 7 + 8 + 9
Using Parentheses ()	n = (1 * 2 * 3 + 4 – 5)
Using Square Brackets []	footballer = ['MESSI', 'NEYMAR', 'SUAREZ']
Using braces {}	X = {1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9}
Using Semicolons (;)	flag = 2; ropes = 3; pole = 4

Python Comments

Comments are used to add remarks or notes in the source code. Comments are not executed by the interpreter. They are added to understand the source code for others. They are used primarily to document the meaning and purpose of source code and its input and output requirements, so as to remember later how it functions and how to use it.

For large and complex software, several programmers are working in teams and sometimes, a programmer has to work on the program written by another programmer. In such situations, documentation in the form of comments is useful to understand the logic of a program.

In Python, we use the hash (#) symbol to start writing a comment. Everything following the # till the end of that line is treated as a comment and the interpreter simply ignores it while executing the statement.

Single line comments

This is single line comment

Multi line comments

This is multi line comment

This is multi line comment

1.5 Python Keywords and Identifiers

In Python, keywords are reserved words and identifiers are the names given to variables or functions.

1.5.1 Python Keywords

Keywords are predefined, reserved words used in Python programming that have special meanings to the compiler.

We cannot use a keyword as a variable name, function name, or any other identifier. They are used to define the syntax and structure of the Python language.

All the keywords except True, False and None are in lowercase and they must be written as they are. The list of all the keywords is given in Table 1.1.

Table 1.2: Python Keywords List

False	await	else	import	pass
None	break	except	in	raise
True	class	finally,	is	return
And	continue	for	lambda	try
As	def	from	nonlocal	while
assert	del	global	not	with
Async	elif	if	or	yield

1.5.2 Python Identifiers

In programming languages, identifiers are names used to identify a variable, function, or other entities in a program.

For example,

language = 'Python'

Here, language is a variable (an identifier) which holds the value 'Python'. We cannot use keywords as variable names as they are reserved names that are built-in to Python. For example,

def = 'Python'

This is wrong to use **def** as a variable name.

1.5.3 Rules for Naming an Identifier

The rules for naming an identifier in Python are as follows:

1. Identifiers are case-sensitive.
2. Keywords cannot be used as identifiers.
3. Identifiers can be combinations of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). However, it must begin with a letter or _. The first letter of an identifier cannot be a digit.
4. We cannot use special symbols like !, @, #, $, %, in the name of an identifier.

5. Identifiers cannot start with a digit. It's a convention to start an identifier with a letter rather _.
6. Identifiers can be combinations of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). We cannot use special symbols like !, @, #, $, %, in the name of an identifier. For example, **1count** is invalid identifier, count1 is a valid identifier.
7. Identifiers can be of any length. However, it is preferred to keep it short and meaningful.

For example, to find the average of marks obtained by a student in three subjects, choose the identifiers as marks1, marks2, marks3 and avg rather than a, b, c, or A, B, C.

avg = (marks1 + marks2 + marks3)/3

Similarly, to calculate the area of a rectangle, use identifier names, such as area, length, breadth instead of single alphabets as identifiers for clarity and more readability.

area = length * breadth

1.5.4 Some Valid and Invalid Identifiers in Python

Table 1.3: Valid and invalid identifiers

Valid Identifiers	Invalid Identifiers
score	@core
return_value	return
highest_score	highest score
name1	1name
convert_to_string	convert to_string

1.6 Python Variables and Literals

1.6.1 Python Variables

In programming, a variable is a container (storage area) to hold data which can be changed later throughout programming. For example,

number = 10

Here, *number* is a variable storing the value 10.

1.6.2 Assigning values to Variables in Python

In Python an assignment statement is used to create new variables and assign specific values to them.

For example, number = 10 is a simple assignment operator that assigns the value 10 on the right to the variable a on the left.

```
>>> # assign value to site_name variable
>>> message = 'Welcome'
>>> print(message)
>>> # Output: Welcome
```

In the above example, we assigned the value 'Welcome' to the '*message*' variable. Then, we printed out the value assigned to the message.

Note: Python is a *type-inferred* language, so you don't have to explicitly define the variable type. It automatically knows that Welcome is a string and declares the 'message' variable as a string.

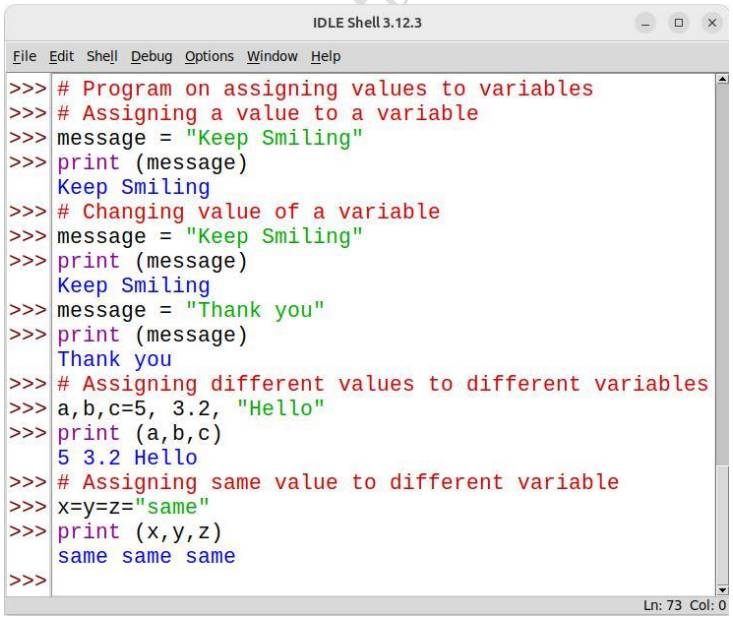
1.6.3 Changing the Value of a Variable in Python

A variable in a program is uniquely identified by a name (identifier). Variable in Python refers to an object — an item or element that is stored in the memory. The value of a variable can be a string (e.g., 'b', 'Global Citizen'), numeric (e.g., 345) or any combination of alphanumeric characters (CD67).

It is possible to change the value of the variable. The various tasks associated with the variable are mentioned below.

Table 1.4: Task associated with the variable

Task	Python Code	Result
Assigning a value to a variable	message = "Keep Smiling" print (message)	Keep Smiling
Changing value of a variable	message = "Keep Smiling" print (message) message = "Thank you" print (message)	Keep Smiling Thank you
Assigning different values to different variables	a,b,c=5, 3.2, "Hello" print(a) print(b) print(c)	5 3.2 "Hello"
Assigning same value to different variable	x=y=z= "Same" print(x) print(y) print(z)	Same Same Same



1.6.4 Rules for Naming Python Variables

1. Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (**A to Z**) or digits (**0 to 9**) or an underscore (_). For example:
2. Create a name that makes sense. For example, *vowel* makes more sense than *v*.
3. If you want to create a variable name having two words, use underscore to separate them. For example:
4. Python is case-sensitive. So *num* and *Num* are different variables. For example,
5. Avoid using keywords like if, True, class, etc. as variable names.

1.7 Python Literals or Constants

Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc. For example, 'Hello, World!', 12, 23.0, 'C', etc.

Literals are also used to assign values to variables or constants. For example,

message = “Keep smiling”

In the above expression, *message* is a variable, and “Keep smiling” is a literal.

The following example illustrates how to declare and print the values of constant.

There are different types of literals in Python. Let's discuss some of the commonly used types in detail.

1.7.1 Python Numeric Literals

Numeric Literals are immutable (unchangeable). Numeric literals can belong to **3** different numerical types: Integer, Float, and Complex.

1. Integer Literals

Integer literals are whole numbers without decimal parts. It also includes negative numbers. For example, 1, -11, 0, 120.

2. Floating-Point Literals

Floating-point literals are numbers that contain decimal parts. Floating-point numbers can also be both positive and negative. For example, 12.5, 8.76, 0.0, -3.14.

3. Complex Literals

Complex literals are numbers that represent complex numbers.

The numerals are in the form a + bj, where a is real and b is imaginary. For example, 6+9j, 2+3j.

1.7.2 Python String Literals

Python string literals are characters used to represent text in code and are defined by enclosing the text within quote marks. Python supports single quote and double quotes formats for string literals. For example,

Single quotes: 'Hello'

Double quotes: "World"

Python Boolean Literals

There are two boolean literals: True and False

```
>>> is_pass = True
>>> print(is_pass)
```

>>> # Output: True

Here, True is a boolean literal assigned to *is_pass*.

1.8 Python Operators

An operator used to perform specific mathematical or logical operation on values are called operands. Operators are special symbols that perform operations on variables and values. For example,

>>> print(5 + 10) # 15

Here, + is an operator that adds two numbers: **5** and **10**.

Python supports several types of operators as discussed below.

Types of Python Operators

Python supports following types operators.

1. Arithmetic Operators
2. Assignment Operators
3. Relational or Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. Special Operators

1.8.1 Arithmetic Operators

Python supports arithmetic operators (+, -, *, /) that are used to perform the four basic arithmetic operations as well as modulus division (%), floor division (//) and exponentiation (**).

Table 2.2 shows various assignment operators available in python

Table 1.5: Arithmetic operators.

Operator	Operation	Meaning	Example
+	Addition	Adds the two numeric values on either side of the operator. It is also used to concatenate two strings.	5 + 2 = 7 'D'+'S' = DS
-	Subtraction	Subtracts the operand on the right from the operand on the left.	4 - 2 = 2
*	Multiplication	Multiplies the two values on both side of the operator. Repeats the item on left of the operator if first operand is a string and second operand is an integer value.	2 * 3 = 6 'D' * 2 = DD
/	Division	Divides the first operand by the second, always returning a floating-point number.	10 / 3 = 3.33 6 / 2 = 3.0
//	Floor Division	Divides the first operand by the second and returns the integer part of the quotient (the largest integer less than or equal to the result).	10 // 3 = 3 6 // 2 = 3 -7 // 2 = 4
%	Modulo	Returns the remainder of the division of the	5 % 2 = 1

		first operand by the second.	
**	Power	Raises the first operand to the power of the second operand.	4 ** 2 = 16

1.8.2 Assignment Operators

Assignment operator assigns or changes the value of the variable on its left. Table 2.3 shows various assignment operators available in python

Table 1.6: Assignment operators

Operator	Operation	Meaning	Example

=	Assignment Operator	Assigns value from right-side operand to left-side operand.	>>> a = 10
			>>> b = a
			>>> b
			10

+=	Addition Assignment	It adds the value of right-side operand to the left-side operand and assigns the result to the left-side operand.	>>> a = 10
			>>> b = 5
			>>> a += b
			15

-=	Subtraction Assignment	It subtracts the value of right-side operand from the left-side operand and assigns the result to left-side operand.	>>> a = 10
			>>> b = 4
			>>> a -= b
			6

*=	Multiplication Assignment	It subtracts the value of right-side operand from the left-side operand and assigns the result to left-side operand.	>>> a = 10
			>>> b = 4
			>>> a *= b
			40

/=	Division Assignment	It multiplies the value of right-side operand with the value of left-side operand and assigns the result to left-side operand.	>>> a = 10
			>>> b = 4
			>>> a /= b
			2.5

%=	Remainder Assignment	It performs modulus operation using two operands and assigns the results to left-side operand.	>>> a=10
			>>> b=3
			>>> a%=b
			>>> a
			1

**=	Exponent Assignment	It performs exponential (power) calculation on operators and assigns value to the left-side operand.	>>> a=5
			>>> b=3
			>>> a**=b
			>>> a
			125

1.8.3 Relational or Comparison Operators

Relational or comparison operator compares the values of the operands on its either side and determines the relationship among them. These are used in decision-making and loops.

Table 1.7:

Operator	Operation	Meaning	Example

==	Is Equal To	If the values of two operands are equal, then the condition is True, otherwise it is False.	>>> a = 3
			>>> b = 5
			>>> a == 5
			False

!=	Not Equal To	If values of two operands are not equal, then condition is True, otherwise it is False.	>>> a = 3
			>>> b = 5
			>>> a != b
			True

>	Greater Than	If the value of the left-side operand is greater than the value of the right- side operand, then condition is True, otherwise it is False.	>>> a = 3
			>>> b = 5
			>>> a > b
			False

<	Less Than	If the value of the left-side operand is less than the value of the right- side operand, then condition is True, otherwise it is False operand, then condition is True, otherwise it is False.	>>> a = 3
			>>> b = 5
			>>> a < b
			True

>=	Greater Than or Equal To	If the value of the left-side operand is greater than or equal to the value of the right-side.	>>> a = 3
			>>> b = 5
			>>> a >= 5
			False

<=	Less Than or Equal To	If the value of the left operand is less than or equal to the value of the right operand, then is True otherwise it is False.	>>> a = 3
			>>> b = 5
			>>> a <= b
			True

1.8.4 Logical Operators

Logical operators are used to check whether an expression is True or False. They are used in decision-making. Every value is logically either True or False. By default, all values are True except None, False, 0 (zero), empty collections "", (), [], {}, and few other special values.

Table 1.8: Logical Operators in Python

Operator	Meaning	Example

and	**Logical AND**: True only if both the operands are True	>>> 10 > 0 and 5 > 0 # True
		>>> 10 > 0 and 5 < 0 # False
		>>> 10 < 0 and 5 > 0 # False
		>>> 10 < 0 and 5 < 0 # False

or	**Logical OR**: True if at least one of the operands is True	>>> 10 > 0 or 5 > 0 # True
		>>> 10 > 0 or 5 < 0 # True
		>>> 10 < 0 or 5 > 0 # True
		>>> 10 < 0 or 5 < 0 # False

not	**Logical NOT**: True if the operand is False and vice-versa. Reverse the logical state of its operand.	>>> not 10 # False >>> not 0 # True

1.8.5 Bitwise Operators

Bitwise operators are the only operators which works on equivalent binary value of the integer operands. First of all, integer operands are converted into binary then respected operator works bit by bit, hence the name is bitwise operators. The result is also converted into decimal format.

The code snippet demonstrated in table 1.6 shows the use of various bitwise operators in Python.

Let x = **10** (0000 1010 in binary) and y = **12** (0000 1100 in binary). The result of bitwise operators on x and y is given below in the table.

Table 1.9: Bitwise operators in Python

Operator	Operation	Meaning	Example

&	Bitwise AND	Returns 1 if both the bits are 1 else 0.
		x = 1010 (10 in decimal)
		y = 1100 (12 in decimal)
		z = 1000 (8 in decimal)
			>>> x = 10
			>>> y = 12
			>>> x & y
			8

|	Bitwise OR	Returns 1 if either of the bit is 1 else 0.
		x = 1010 (10 in decimal)
		y = 1100 (12 in decimal)
		z = 1110 (14 in decimal)
			>>> x = 10
			>>> y = 12
			>>> x | y
			14

~	Bitwise NOT	Returns one's complemented of the number.
		z = ~a
		= ~ 1010
		= - (1010 + 1)
		= - 11
			>>> x = 10
			>>> z = ~x
			>>> z
			-11

^	Bitwise XOR	Returns 1 if one of the bits is 1 and the other is 0 else returns false.
		X = 1010 (10 in decimal)
		y = 1100 (12 in decimal)
		z = 0110 (6 in decimal)
			>>> x=10
			>>> y=12
			>>> z=x^y
			>>> z
			6

>>	Bitwise right shift	Shifts the bits of the number to the right and fills 0 on voids left (fills 1 in the case of a negative number) as a result. Similar effect as of dividing the number with	>>> x = 10
			>>> x >> 2
			2

		some power of two.	

<<	Bitwise left shift	Shifts the bits of the number to the left and fills 0 on voids right as a result. Similar effect as of multiplying the number with some power of two.	>>> x = 10
			>>> x << 2
			40

1.8.6 Special Operators

Python language offers some special types of operators like the identity operator and the membership operator. They are described below with examples.

Identity operators

Identity operators are used to determine whether the value of a variable is of a certain type or not. Identity operators can also be used to determine whether two variables are referring to the same object or not. There are two identity operators explained in table 10.4.

Table 1.10 Identity operators in Python

Operator	Description	Example

is	Evaluates True if the variables on either side of the operator point towards the same memory location and False otherwise. var1 is var2 results to True if id(var1) is equal to id(var2).	>>> num1=10
		>>> num2=5
		>>> num3=num1
		>>> num2 is num1
		False
		>>> num3 is num1
		True

is not	Evaluates to False if the variables on either side of the operator point to the same memory location and True otherwise. var1 is not var2 results to True if id(var1) is not equal to id(var2).	>>> num1=10
		>>> num2=5
		>>> num3=num1
		>>> num2 is not num1
		True
		>>> num3 is not num1
		False

Membership Operators

Membership operators are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary). In a dictionary, we can only test for the presence of a key, not the value.

There are two membership operators explained in Table 1.11.

Table 1.11: Membership operators in Python

Operator	Description	Example (Try in Lab)

in	Returns True if the variable/value is found in the specified sequence and False otherwise.	>>> a = [10,20,30,40,50]
		>>> 10 in a
		True
		>>> 25 in a

		False

not in	Returns True if the variable/value is not found in the specified sequence and False otherwise.	>>> a = [10,20,30,40,50]
		>>> 10 not in a
		False
		>>> 25 not in a
		True

1.9 Expressions

An expression is defined as a combination of constants, variables, and operators. An expression always evaluates to a value. A value or a standalone variable is also considered as an expression but a standalone operator is not an expression. Some examples of valid expressions are given below.

100

num

num – 20.4

+ 3.14

23/3 -5 * 7(14 -2)

"Global" + "Citizen"

1.9.1 Precedence of Operators

Evaluation of the expression is based on precedence of operators. When an expression contains different types of operators, precedence determines which operator should be applied first. The higher precedence operator is evaluated before the lower precedence operator. Most of the operators studied till now are binary operators with two operands. The unary operators need only one operand, and they have a higher precedence than the binary operators. The minus (-) as well as + (plus) operators can act as both unary and binary operators, but "not" is a unary logical operator.

#Depth is using - (minus) as unary operator

Value = -Depth

#not is a unary operator, negates True print (not (True))

Table 1.11 lists precedence of all operators from highest to lowest.

Table 1.12: Precedence of all operators in Python

Precedence	Operators	Description
1	**	Exponentiation (power)
2	~ ,+, -	Complement, unary plus and unary minus
3	* ,/, %, //	Multiply, divide, modulo and floor division
4	+, -	Addition and subtraction
5	<= ,< ,> ,>=	Relational operators
6	== ,!=	Equality operators

7	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
8	is, is not	Identity operators
9	in, not in	Membership operators
10	not, or, and	Logical operators

Parenthesis can be used to override the precedence of operators. The expression within () is evaluated first. For operators with equal precedence, the expression is evaluated from left to right.

Example 1

How will Python evaluate the following expression? 20 + 30 * 40

Solution:

= 2 (30 * 40) #Step 1

precedence of * is more than that of +

= 20 + 1200 #Step 2

= 1220 #Step 3

Example 2

How will Python evaluate the following expression? 20 – 30 + 40

Solution:

The two operators (–) and (+) have equal precedence. Thus, the first operator, i.e., subtraction is applied before the second operator, i.e., addition (left to right).

= (20 – 30) + 40 #Step 1

= - 10 + 40 #Step 2

= 30 #Step 3

Example 3

How will Python evaluate the following expression? (20 + 30) * 40

Solution:

= (20 + 30) * 40 # Step 1

#using parenthesis (), we have forced precedence of + to be more than that of *

= 50 * 40 # Step 2

= 2000 # Step 3

Example 4

How will the following expression be evaluated in Python? 15.0 / 4 + (8 + 3.0)

Solution:

15.0 / 4 + (8.0 + 3.0 #Step 1

15.0 / 4.0 + 11.0 #Step 2

3.75 + 11.0 #Step 3

14.75 #Step 4

1.10 Python Basic Input and Output

1.10.1 Python Output

In Python, the print() function is used to print the output. For example,

print('Hello World')

Output: Hello World

The print() function displays the string enclosed inside the single quotation.

Syntax of print()

In the above code, the print() function is taking a single parameter. However, the actual syntax of the print function accepts 5 parameters.

print(object= separator= end= file= flush=)

Where,

object - value(s) to be printed

sep (optional) - allows us to separate multiple **objects** inside print().

end (optional) - allows us to add add specific values like new line "\n", tab "\t"

file (optional) - where the values are printed. Its default value is sys.stdout (screen)

flush (optional) - boolean specifying if the output is flushed or buffered. Default: False

Example 1: Python Print Statement

>>> print("Hello")

Hello

>>> print('World')

World

In the above example, the print() statement only includes the object to be printed. Here, the value for **end** is not used. Hence, it takes the default value '\n'. So we get the output in two different lines.

Example 2: Python print() with end Parameter

>>> # print with end whitespace

>>> print(“Hello”, end= “ “)

>>> print(“World!”)

Output

Hello World!

Notice that the end= “ “ after the end of the first print() statement, print the output in a single line separated by space.

Example 3: Python print() with sep parameter

>>> print(“Hello”, “World!”, sep= '. ')

Output

Hello.World!

In the above example, the print() statement includes strings separated by a comma. Notice that the optional parameter sep= ". " inside the print() statement, print the output separated by . not comma.

Example: Print Concatenated Strings

We can also join two strings together inside the print() statement. For example,

print("Hello" + "World!")

Output : HelloWorld

Output formatting

To make the output look attractive, the output can be formatted using the str.format() method. For example,

>>> x = 5

>>> y = 10

>>> print('The value of x is {} and y is {}'.format(x,y))

Output

The value of x is 5 and y is 10

1.10.2 Python Input

While programming, you may require to take the input from the user. In Python, the input () function prompts the user to enter data. It accepts all user input as string. The user may enter a number or a string but the input () function treats them as strings only. The syntax for input statement is:

Syntax of input()

input ([Prompt])

Prompt is the string we may like to display on the screen prior to taking input, and it is optional. When a prompt is specified, first it is displayed on the screen after which the user can enter data. The input() takes exactly what is typed from the keyboard, converts it into a string and assigns it to the variable on the left-hand side of the assignment operator (=). Entering data for the input function is terminated by pressing the enter key.

Example 1: Taking User Input

```
# using input() to take user input
fname = input("Enter your first name : ")
Enter your first name : Aadi
age = input("Enter your age : ")
Enter your age : 17
print (fname, type(fname))
print (age, type(age))

Output

Aadi <class 'str'>
17 <class 'str'>
```

In Example 1.2, the variable fname will get the string 'Aadi', entered by the user. Similarly, the variable age will get the string '17'. We can typecast or change the datatype of the string data accepted from the user to an appropriate numeric value. The code in Example 1.3 converts the accepted string to an integer. If the user enters any non-

numeric value, an error will be generated.

Example 2

```
# Function int() to convert string to integer
age = int(input("Enter your age : "))
Enter your age : 17
print (age, type(age))
```

Output

17 <class 'int'>

```
File  Edit  Shell  Debug  Options  Window  Help
>>> # using input() to take user input
>>> fname = input("Enter your first name : ")
    Enter your first name : Aadi
>>> age = input("Enter your age : ")
    Enter your age : 17
>>> print (fname, type(fname))
    Aadi <class 'str'>
>>> print (age, type(age))
    17 <class 'str'>
>>>
>>> # Function int() to convert string to integer
>>> age = int(input("Enter your age : "))
    Enter your age : 17
>>> print (age, type(age))
    17 <class 'int'>
>>>
                                                Ln: 185  Col: 24
```

1.11 Data Types

Every value belongs to a specific data type in Python. Data type identifies the type of data values a variable can hold and the operations that can be performed on that data. Figure 1.8 enlists the data types available in Python.

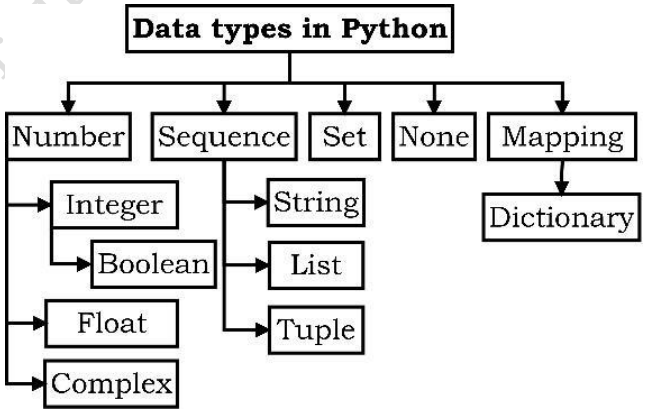


Figure 1.8: Data types in Python

1.11.1 Number

Number data type stores numerical values only. It is further classified into three different types: int, float and complex. Boolean data type (bool) is a subtype of integer. It is a unique data type, consisting of two constants, True and False. Boolean True value is

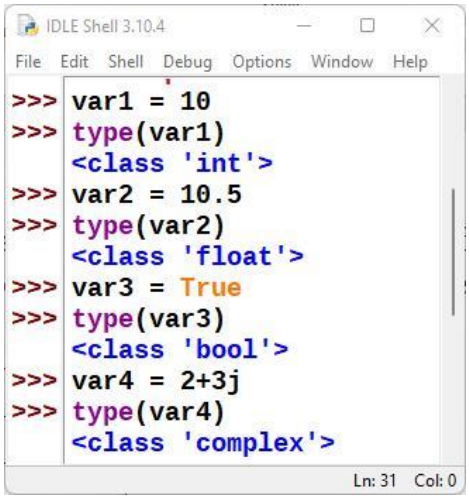
non-zero, non-null and non-empty. Boolean False is the value zero.

Table 1.13 Numeric data types

Type/Class	Description	Examples
int	integer numbers	–12, –3, 0, 125, 2
float	real or floating-point numbers	–2.04, 4.0, 14.23
complex	complex numbers	3 + 4i, 2 – 2i
bool	Boolean values	

Let us execute a few statements in interactive mode of Python IDLE to determine the data type of the variable using built-in function type ().

Example 1



Variables of simple data types like integer, float, boolean that hold single value. But such variables are not useful to hold a long list of information, such as months in a year, student names in a class, names and numbers in a phone book or the list of artifacts in a museum. For this, Python provides other data types like *tuples, lists, dictionaries* and *sets.*

1.11.2 Sequence

A Python sequence is an ordered collection of items, where each item is indexed by an integer. The three types of sequence data types available in Python are *Strings, Lists* and *Tuples*. We will learn about each of them in detail in later Sessions. A brief introduction to these data types is as follows:

String **–** String is a group of characters. These characters may be alphabets, digits or special characters including spaces. String values are enclosed either in single quotation marks (e.g., 'Hello') or in double quotation marks (e.g., "Hello"). The quotes are not a part of the string; they are used to mark the beginning and end of the string for the interpreter. For example,

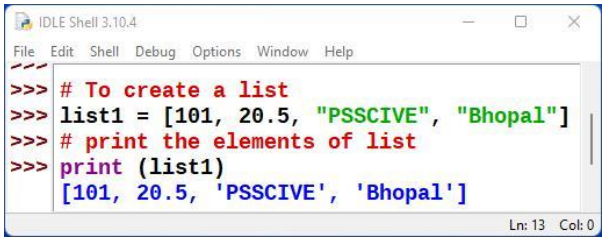
```
>>> str1 = 'Hello Friend'
>>> str2 = "452"
```

It is not possible to perform numerical operations on strings, even when the string

contains a numeric value, as in str2.

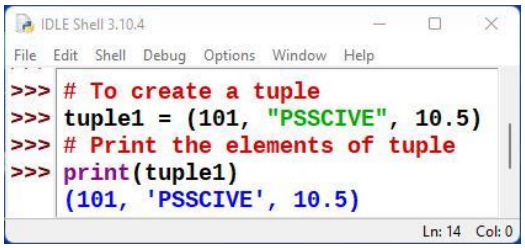
List **–** List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

Example 2



Tuple **–** Tuple is a sequence of items separated by commas and items are enclosed in parenthesis (). This is unlike list, where values are enclosed in brackets []. Once created, we cannot change the tuple.

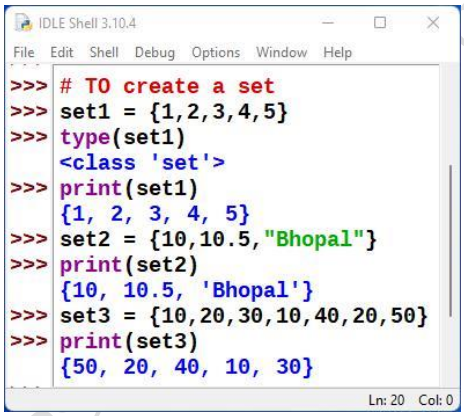
Example 3



1.11.3 Set

Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets {}. A set is similar to a list, except that it cannot have duplicate entries. Once created, elements of a set cannot be changed.

Example 4

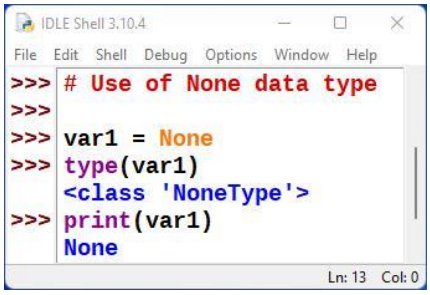


In the above example, set1 is a collection of 5 integers. The set2 is a collection of different data types of elements. You must have noticed that elements of set2 have been displayed in an order different from the order in which they have entered. The reason for this is that the set is unordered. If you run the same code again, it is possible that you will get an output with the elements arranged in a different order. A set does not allow duplicate values, that you may observe for set3. Here, all the duplicate values have been removed from set3.

1.11.4 None

None is a special data type with a single value. It is used to signify the absence of value in a situation. None supports no special operations, and it is neither False nor 0 (zero).

Example 5



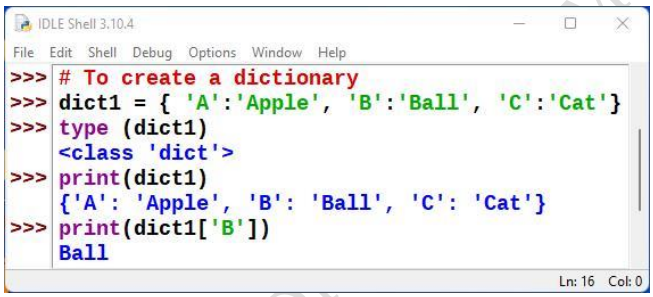
1.11.5 Mapping

Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called a dictionary.

Dictionary

Dictionaries in Python hold data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key: value pairs of a dictionary can be accessed using the key. The keys are usually strings and their values can be any data type. In order to access any value in the dictionary, we have to specify its key in square brackets [].

Example 6



1.12 Mutable and Immutable Data Types

Sometimes we may require to change or update the values of certain variables used in a program. However, for certain data types, Python does not allow to change the values once a variable of that type has been created and values are assigned.

Variables whose values can be changed after they are created and assigned are called mutable. Variables whose values cannot be changed after they are created and assigned are called immutable. When an attempt is made to update the value of an immutable variable, the old variable is destroyed and a new variable is created by the same name in memory. Python data types can be classified into mutable and immutable as shown in Figure 1.9.

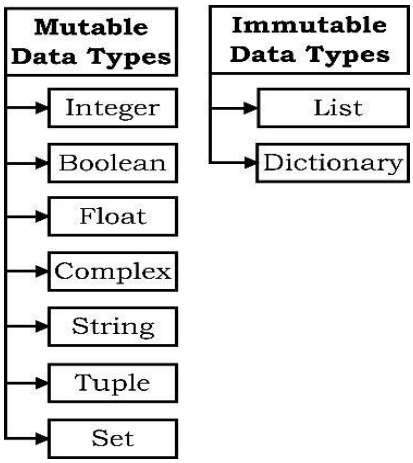


Fig. 1.9: Classification of data types

Let us now see what happens when an attempt is made to update the value of a variable. >>> num1 = 300 This statement will create an object with value 300 and the object is referenced by the identifier num1 as shown in Figure 1.10.

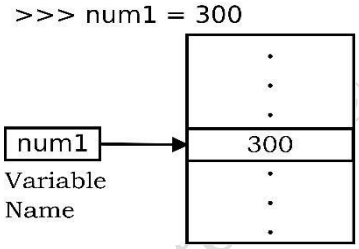


Fig. 1.10: Object and its identifier

The statement num2 = num1 will make num2 refer to the value 300, also being referred by num1, and stored at memory location number, say 1000. So, num1 shares the referenced location with num2 as shown in Figure 1.11.

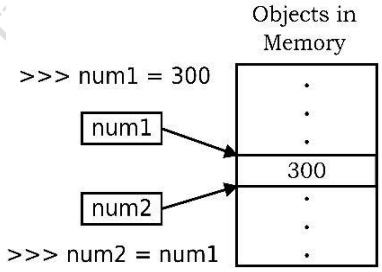


Fig. 1.11 Variables with same value have same identifier

In this manner Python makes the assignment effective by copying only the reference, and not the data:

>>> num1 = num2 + 100

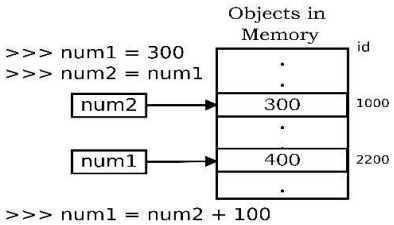


Fig. 1.12: Variables with different values have different identifiers

This statement *num1 = num2 + 100* links the variable num1 to a new object stored at memory location number say 2200 having a value 400. As num1 is an integer, which is an immutable type, it is rebuilt, as shown in Figure 1.12.

Python Type Conversion

In programming, type conversion is the process of converting data of one type to another. For example: converting int data to str.

There are two types of type conversion in Python.

1. ***Implicit Conversion*** – automatic type conversion
2. ***Explicit Conversion*** – manual type conversion

Python Implicit Type Conversion

In certain situations, Python automatically converts one data type to another. This is known as implicit type conversion.

Example 1: Converting integer to float

Let's see an example where Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

```
# Example of implicit conversion
quantity = 10           # quantity in integer
cost_per_item = 10.50   # Cost of item in float
total_cost = quantity * cost_per_item
print("Total Cost:", total_cost, type(total_cost))
```

Output

```
Total Cost: 105.0 <class 'float'>
```

In the above example, although the data type of quantity is int and cost_per_item is of float type. The total_cost is the product of quantity and cost_per_item resulted into float data type. This is an example of implicit type conversion.

It is because Python always converts smaller data types to larger data types to avoid the loss of data.

But if you try to add str and int you get TypeError. For example, “10 item” * 10.50, Python is not able to use Implicit Conversion in such conditions. Python has a solution for these types of situations which is known as Explicit Conversion.

Explicit Type Conversion

In Explicit Type Conversion, users convert the data type of an object to required data type. The built-in functions like int(), float(), str(), etc are used to perform explicit type conversion.

This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

```
# Example of explicit conversion
cost_per_item = 10  # Cost of item in int
quantity = input("Enter quantity: ") # User input is always string
```

```
print("Data type of quantity before Type Casting:",type(quantity))
quantity = int(quantity)
print("Data type of quantity after Type Casting:",type(quantity))
total_cost = cost_per_item * quantity
print("Total Cost :", total_cost, type(total_cost))
print("Total Cost :", total_cost, type(total_cost))
```

Output

```
Enter quantity: 5
Data type of quantity before Type Casting: <class 'str'>
Data type of quantity after Type Casting: <class 'int'>
Total Cost : 50 <class 'int'>
```

In the above example, even if the user enters the quantity as 5, it is initially stored as the string "5" because the user input is always a string.

The int() function explicitly converts the string quantity into integer for numerical calculations.

If we try to calculate the total cost without converting the string value of quantity to integer, it will raise a ValueError.

Usage of Python Data Types

Each data type has some specific properties. Appropriate use of data type is dependent on the situation. It is preferred to use *lists* for a simple iterative collection of data that may go for frequent modifications. For example, if we store the names of students of a class in a list, then it is easy to update the list when some new students join or some leave the course. *Tuples* are used when we do not need any change in the data. For example, names of months in a year. When we need uniqueness of elements and to avoid duplicity it is preferable to use *sets*, for example, list of artifacts in a museum. If our data is being constantly modified or we need a fast lookup based on a custom key or we need a logical association between the key: value pair, it is advised to use *dictionaries*. A mobile phone book is a good dictionary application.

Summary

- Type Conversion is the conversion of an object from one data type to another data type.
- Implicit Type Conversion is automatically performed by the Python interpreter.
- Python avoids the loss of data in Implicit Type Conversion.
- Explicit Type Conversion is also called Type Casting, the data types of objects are converted using predefined functions by the user.
- In Type Casting, loss of data may occur as we enforce the object to a specific data type.

CHECK YOUR PROGRESS

A. Multiple Choice Questions

1. Which of the following is not a Python keyword? (a) if (b) else (c) elif (d) eval
2. The keyword True and False represent: (a) String values (b) Integer values (c) Boolean values (d) Floating-point values
3. Which of the following is a valid variable name in Python? (a) 1_variable (b) my-variable (c) _value (d) for
4. What symbol is used to assign a value to a variable in Python? (a) == (b) = (c) := (d) +=
5. In Python, variable names are: (a) Case-insensitive (b) Case-sensitive (c) Type-sensitive (d) Length-sensitive
6. Which of the following is not a valid rule for naming variables in Python? (a) Variable names must start with a letter or an underscore (b) Variable names can contain alphanumeric characters and underscores (c) Variable names can start with a number (d) Variable names cannot be a Python keyword.
7. What is an identifier in Python? (a) A reserved word with special meaning (b) A name given to a variable, function, class, module, etc (c) A built-in function like print() (d) A data type like int or str.
8. Which of the following is an invalid identifier in Python? (a) _my_var (b) 2ndVar (c) var_name (d) myVar
9. Which operator is used to calculate the remainder of a division in Python? (a) // (b) / (c) % (d) **
10. What is the output of 10 // 3 in Python? (a)3.33 (b) 3 (c) 1 (d) 4
11. Which of the following is the equality operator in Python? (a) = (b) != (c) == (d) >=
12. What is the result of 5 > 3 and 2 < 4? (a) True (b) False (c) Error (d) None
13. Which operator checks if two variables refer to the same object in memory? (a) == (b) is (c) in (d) !=
14. Which of the following expressions is equivalent to x = x + 5? (a) x += 5 (b) x -= 5 (c) x *= 5 (d) x //= 5
15. What is the purpose of the in operator? (a) To perform addition (b) To check if a value exists in a sequence (c) To assign a value to a variable (d) To compare two numbers
16. What is the output of print(2 ** 3)? (a) 5 (b) 6 (c) 8 (d) 9
17. Which of the following is a bitwise operator? (a) + (b) == (c) & (d) not
18. Which function is used to display output on the console in Python? (a) display() (b) show() (c) print() (d) output()
19. What is the default separator used between multiple values when printed using the print() function? (a) Comma (,) (b) Hyphen (-) (c) Space ( ) (d) Newline (\n)
20. What is the output of print("Hello", "World", sep="—")? (a) Hello World (b) Hello—World (c) HelloWorld (d) Hello, World

21. What is the default end character in the print() function? (a) Space ( ) (b) Tab (\t) (c) Newline (\n) (d) Period (.)
22. What is the output of print("Python", end="@"); print("is fun")? (a) Python is fun (b) Python@is fun (c) Python@is fund) (d) Python@ is fun
23. How can you redirect the output of a print() statement to a file instead of the console? (a) Using the output() function (b) Specifying the file parameter in print() (c) Using the write_to_file() function (d) Python doesn't support redirecting print() output to files
24. What is the output of the expression 4 + 3 % 5? (a) 7 (b) 2 (c) 4 (d) 1
25. According to operator precedence, which operation is performed first in the expression 10 - 4 * 2? (a) Subtraction (-) (b) Multiplication (*) (c) Both have equal precedence (d) Depends on the order of appearance

B. Fill in the Blanks

1. Python programs/scripts are stored in files with the . _____ extension.
2. In Python, the _____ operator is used to assign a value to a variable.
3. _____ are the user-defined names for different parts of a program like variables, functions, etc.
4. Python is a _____-sensitive language.
5. The _____ function is used to take input from the user.
6. The input() function returns the user's input as a _____ data type by default.
7. To convert a string input to an integer, the _____() function is used.
8. The _____ function is used to display output on the console.
9. Comments in Python begin with the _____ character.
10. The _____ operator is used to calculate the remainder of a division.
11. The _____ operator is used for exponentiation.
12. The _____ statement is used to create a new variable and assign a value.
13. The _____ statement skips the current iteration of a loop and proceeds with the next iteration.
14. An _____ is a combination of values, variables, and operators that evaluates to a single result.
15. Python allows variable names of _____ length.
16. In Python, _____ are reserved words that have special meaning and purpose.
17. The process of manually changing the data type of a value using functions like int(), float(), or str() is called _____ conversion or type _____.
18. In the expression 10 + 5 * 2, the _____ operator is evaluated before the _____ operator.
19. The result of the expression True and False is _____.
20. The result of the expression True or False is _____.
21. To convert a string input to an integer, the _____() function is used.

22. The _____ statement is used to create a new variable and assign a value.
23. The _____ function is used to take input from the user.
24. Python supports _____ and _____ type conversion in expressions.
25. If you try to convert a non-numeric string to an integer using int(), Python will raise a _____Error.

C. State whether True or False

1. Indentation is used for code readability but doesn't affect program execution in Python.
2. You must explicitly declare the data type of a variable before using it in Python.
3. The input() function in Python returns the user's input as an integer by default.
4. The print() function automatically adds a newline character at the end of its output by default.
5. Python is a platform-independent language.
6. Keywords can be used as identifier names in Python.
7. The + operator can be used to concatenate two strings in Python.
8. The == operator checks if two variables refer to the exact same object in memory.
9. Operator precedence determines the order in which operators are evaluated in an expression.
10. Python does not support compound assignment operators like += or -=

D. Programming Questions

1. Which of the following identifier names are invalid and why?

   Serial no.

   1st_Room

   Hundred$

   Total Marks

   total Marks

   total-Marks

   _Percentage

2. Write the Python assignment statements to:

   I. Assign 10 to variable length and 20 to variable breadth.

   II. Assign the average of values of variable length and breadth to a variable sum.

   III. Assign a list containing strings 'Paper', 'Gel Pen', and 'Eraser' to a variable stationery.

   IV. Assign strings 'Mohandas', 'Karamchand', and 'Gandhi' to variables first, middle and last.

   V. Assign the concatenated value of string variables first, middle and last to variable full name. Make sure to incorporate blank spaces appropriately between different

parts of the names.

3. Write logical expressions corresponding to the following statements in Python and evaluate the expressions, assuming variables num1, num2, num3, first, middle, last are already having meaningful values:

   a) The sum of 20 and –10 is less than 12.

   num3 is not more than 24.

   6.75 s between the values of integers num1 and num2.

   b) The string 'middle' is larger than the string 'first' and smaller than the string 'last'.

   c) List Stationery is empty.

4. Add a pair of parentheses to each expression so that it evaluates to True.

   a) 0 == 1 == 2

   b) 2 + 3 == 4 + 5 == 7

   c) 1 < -1 == 3 > 4

5. Write the output of the following:

   a) num1 = 4

   num2 = num1 + 1

   num1 = 2 print (num1, num2)

   b) num1, num2 = 2, 6

   num1, num2 = num2, num1 + 2

   print (num1, num2)

   c) num1, num2 = 2, 3

   num3, num2 = num1, num3 + 1

   print (num1, num2, num3)

6. Which data type will be used to represent the following data values and why?

Data values	**Data type**	**Reason**
Number of months in a year		
Resident of Delhi or not		
Mobile number		
Pocket money		
Volume of a sphere		
Perimeter of a square		
Name of the student		
Address of the student		

7. What will be the output of statement print(num1) in the following example when num1 = 4, num2 = 3, num3 = 2

   num1 += num2 + num

   num1 = num1 ** (num2 + num3)

   num1 **= num2 + num3

num1 = '5' + '5'

num1 = 2+9*((3*12)-8)/10

num1 = 24 // 4 // 2

num1 = float(10)

num1 = int('3.14')

8. What will be the output of following statements
    3. print('Bye' == 'BYE')
    4. print(10 != 9 and 20 >= 20)
    5. print(10 + 6 * 2 ** 2 != 9//4 -3 and 29 >= 29/9)
    6. print(5 % 10 + 10 < 50 and 29 <= 29)
    7. print((0 < 6) or (not(10 == 6) and 10<0)))
9. Write a Python program to convert temperature in degree Celsius to degree Fahrenheit. If water boils at 100°C and freezes at 0°C, use the program to find out what is the boiling point and freezing point of water on the Fahrenheit scale. (Hint: T(°F) = T(°C) × 9/5 + 32)
10. Write a Python program to calculate the amount payable if money has been lent on simple interest. Principal or money lent = P, Rate of interest = R% per annum and Time = T years. Then Simple Interest (SI) = (P ×R × T)/ 100. Amount payable = Principal + SI. P, R and T are given as input to the program.
11. Write a program to calculate in how many days a work will be completed by three persons A, B and C together. A, B, C take x days, y days and z days respectively to do the job alone. The formula to calculate the number of days if they work together is xyz/(xy + yz + xz) days where x, y, and z are given as input to the program.
12. Write a program to enter two integers and perform all arithmetic operations on them.
13. Write a program to enter five subject marks and print the average marks.
14. Write a program to swap two numbers using a third variable.
15. Write a program to swap two numbers without using a third variable.
16. Write a program to repeat the string "GOOD MORNING" *n* times, *n* is integer entered by the user.
17. Write a program to find the average of three numbers.
18. Write a Python program to find the volume of spheres with radius 7cm, 12cm, 16cm, respectively. The volume of a sphere with radius r is $4/3\pi r^3$.
19. Write a program that asks the user to enter their name and age. Print a message addressed to the user that tells the user the year in which they will turn 100 years old.
20. The formula $E = mc^2$ states that the equivalent energy (E) can be calculated as the mass (m) multiplied by the speed of light (c = about 3×10 m/s) squared. Write a program that accepts the mass of an object and determines its energy.

Session 2. Control Flow in Python

2.1 Python if...else Statement

In computer programming, the **if statement** is a conditional statement. It is used to execute a block of code only when a specific condition is true.

For example, suppose we need to assign different grades to students based on their scores.

1. If a student scores above **90**, assign grade **A**
2. If a student scores above **75**, assign grade **B**
3. If a student scores above **60**, assign grade **C**

This tasks can be achieved using the if statement.

2.1.1 Python if Statement

An if statement executes a block of code only when the specified condition is true.

Syntax

```
if condition:
    # body of if statement
```

Here, **condition** is a boolean expression, such as number > 5, that evaluates to either True or False.

- If condition evaluates to True, the body of the if statement is executed.
- If condition evaluates to False, the body of the if statement will be skipped from execution.

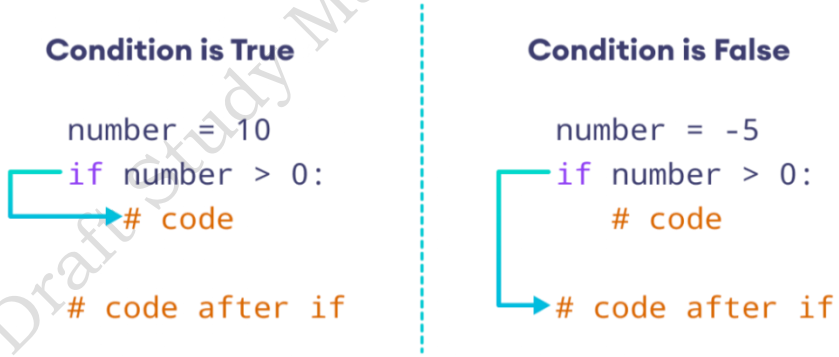


Fig. 2.1 Working of if Statement

Example: Python if Statement

```
# Example of Python if Statement
number = int(input('Enter a number: '))
# check if number is greater than 0
if number > 0:
    print(f'{number} is a positive number.')
# End of if statement
print('A statement next to if statement.')
```

Output

```
Enter a number: 9
9 is a positive number.
A statement next to if statement.

Enter a number: -7
A statement next to if statement.
```

If the user enters **9**, the condition number > 0 evaluates to True. Therefore, the body of if is executed.

If the user enters **-7**, the condition number > 0 evaluates to False. Therefore, the body of if is skipped from execution.

2.1.2 Indentation in Python

In Python, indentation refers to the whitespace (spaces or tabs) at the beginning of a line of code. Python uses indentation to define a block of code, such as the body of an if statement. For example,

In the above example, the statement "print(f'{number} is a positive number.')" is in the body of the if statement, because of indentation. The statement "print('A statement next to if statement.')" is out of the if statement. If you don't give indentation immediately below the if statement, the code gives an error of indentation.

2.1.3 Python if...else Statement

An if statement can have an optional else clause. The else statement executes if the condition in the if statement evaluates to False.

Syntax

```
if condition:
    # body of if statement
else:
    # body of else statement
```

Here, if the condition inside the if statement evaluates to

- **True** - the body of if executes, and the body of else is skipped.
- **False** - the body of else executes, and the body of if is skipped

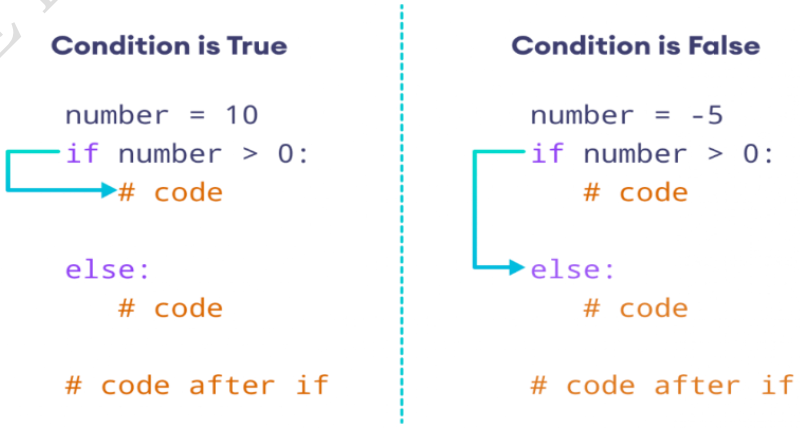


Fig. 2.2 Working of if...else Statement

Example: Python if...else Statement

```
# Example of Python if…else Statement
number = int(input('Enter a number: '))
if number > 0:
    print(f'{number} is a positive number.')
else:
    print(f'{number} is a negative number.')
print('Execute this statement next to if...else')
```

Output

```
Enter a number: 9
9 is a positive number.
Execute this statement next to if...else
Enter a number: -5
-5 is a negative number.
Execute this statement next to if...else
```

If the user enters **9**, the condition number > 0 evaluates to True. Therefore, the body of if is executed and the body of else is skipped.

If the user enters **-5**, the condition number > 0 evaluates to False. Therefore, the body of if is skipped and the body of else is executed.

2.1.4 Python if...elif...else Statement

The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...elif...else statement.

Syntax

```
if condition1:
    # code block 1
elif condition2:
    # code block 2
else:
    # code block 3
```

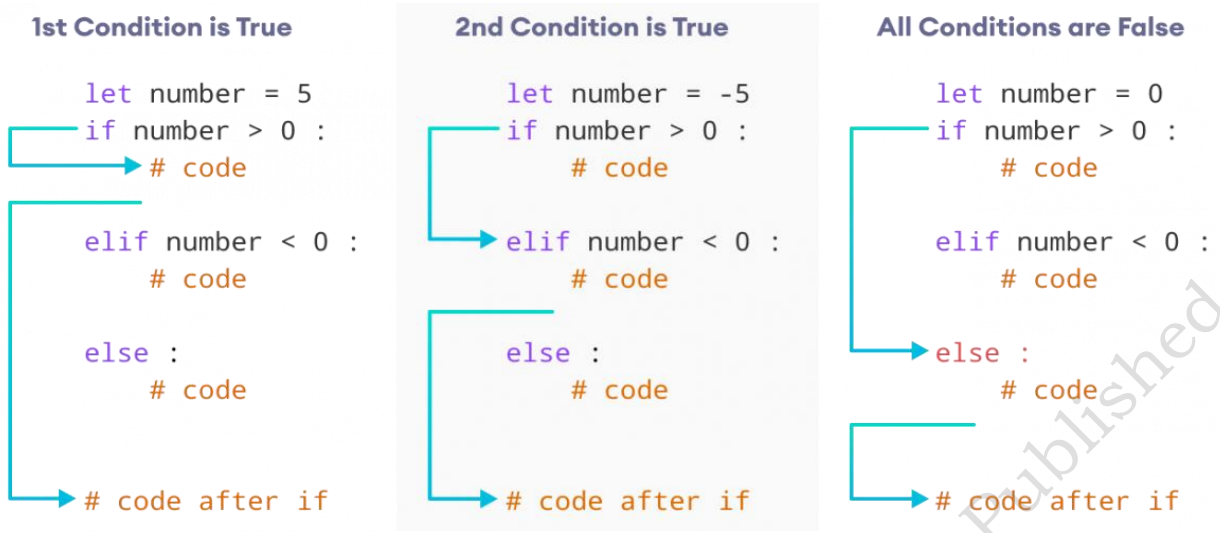


Fig. 2.3 Working of if...elif...else Statement

Example: Python if...elif...else Statement

```
# Example of Python if...elif...else Statement
number = int(input('Enter a number: '))
if number > 0:
    print(f'{number} is a positive number.')
elif number < 0:
    print(f'{number} is a negative number.')
else:
    print(f'{number} is Zero.')
print('Execute this statement next to if...elif...else')
```

Output

```
Enter a number: 10
10 is a positive number.
Execute this statement next to if...elif...else

Enter a number: -5
-5 is a negative number.
Execute this statement next to if...elif...else

Enter a number: 0
0 is Zero.
Execute this statement next to if...elif...else
```

If the user enters **10**, the condition number > 0 evaluates to True. Therefore, the body of if is executed and the body of elif...else is skipped.

In the second run, the user enters **-5**, the condition number > 0 evaluates to False. Therefore, the condition in the statement elif is checked. The condition number < 0 evaluates to True. Therefore, the body of elif is executed and the body of else is skipped.

In the next run, the user enters **0**, the condition number > 0 evaluates to False. Then the condition in the statement elif is checked. The condition number < 0 evaluates to False. Therefore, the body of someone else is executed.

2.1.5 Python Nested if Statements

It is possible to include an if statement inside another if statement. For example,

```
# Example of Python Nested if Statement
number = int(input('Enter a number: '))
# outer if statement
if number >= 0:
    # inner if statement
    if number == 0:
      print(f'{number} is Zero.')
    # inner else statement
    else:
        print(f'{number} is a positive number.')
# outer else statement
else:
    print(f'{number} is a negative number.')
```

Output

```
Enter a number: 5
5 is a positive number.

Enter a number: 0
0 is Zero.

Enter a number: -5
-5 is a negative number.
```

Here's how this program works.

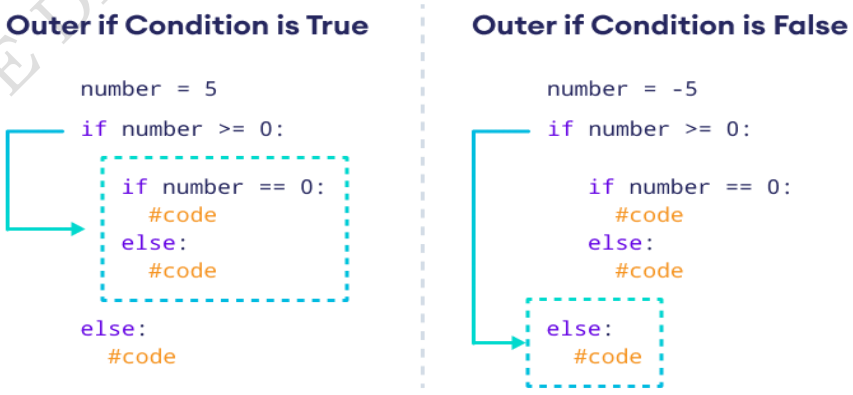


Fig. 2.4 Working of Nested if Statement

More on Python if...else Statement

Compact if Statement

In certain situations, the if statement can be simplified into a single line. For example,

```
number = 10
if number > 0: print('Positive')
```

This one-liner approach retains the same functionality but in a more concise format.

Ternary Operator in Python if...else

Python doesn't have a ternary operator. However, we can use if...else to work like a ternary operator in other languages. For example,

```
grade = 40
result = 'pass' if number >= 50 else 'fail'
print(result)
```

Logical Operators to Add Multiple Conditions

If needed, we can use logical operators such as and and or to create complex conditions to work with an if statement.

```
# Using Logical Operators for Multiple Conditions
age = int(input('Enter your age: '))
nationality = str(input('Enter your nationality: '))
# add two conditions using and operator
if age >= 18 and nationality == 'indian':
    print('Eligible for voting in India.')
else:
    print('Not eligible for voting in India')

Output
Enter your age: 17
Enter your nationality: indian
Not eligible for voting in India

Enter your age: 18
Enter your nationality: indian
Eligible for voting in India.

Enter your age: 19
Enter your nationality: in
Not eligible for voting in India
```

Here, we used the logical operator and to add two conditions in the if statement.

We also used >= (comparison operator) to compare two values.

Logical and comparison operators can be used with if...else statements.

2.2 Python for Loop

In Python, a for loop is used to iterate over sequences such as lists, strings, dictionaries, etc. For example,

```
subjects = ['English','Physics','Chemestry','Maths']
# access elements of the list one by one
for sub in subjects:
    print(sub)

Output
Physics
Chemestry
Maths
```

In the above example, a list named *subjects* is created. Since the list has three elements, the loop iterates **3** times.

The value of sub is

1. Physics in the first iteration.
2. Chemistry in the second iteration.
3. Maths in the third iteration.

for loop Syntax

```
for val in sequence:
    # run this code
```

The for loop iterates over the elements of **sequence** in order, and in each iteration, the body of the loop is executed.

The loop ends after the body of the loop is executed for the last item.

Indentation in Loop

In Python, we use indentation (spaces at the beginning of a line) to define a block of code, such as the body of a loop. For example,

```
subjects = ['Physics','Chemestry','Maths']
# start of the loop
for sub in subjects:
    print(sub)
    print('-----------')
# end of the for loop
print('Last statement')

Output

Physics
-----------
Chemestry
-----------
Maths
-----------
```

```
Last statement
```

Here, print('Last statement') is outside the body of the loop. Therefore, this statement is executed only once at the end.

Example: Loop Through a String

Iterating through a string, you will get individual characters of the string one by one.

```
subject = 'Maths'
# iterate over each character in subject
for x in subject:
    print(x)

Output
M
a
t
h
s
```

Here, each character of the string *subect* is printed using a for loop.

for Loop with Python range ()

In Python, the range() function returns a sequence of numbers. For example,

generate numbers from 0 to 3

values = range(0, 4)

Here, range(0, 4) returns a sequence of **0**, **1**, **2** ,and **3**.

Since the range() function returns a sequence of numbers, we can iterate over it using a for loop. For example,

```
# iterate from i = 0 to i = 3
for i in range(0, 4):
    print(i)

Output
0
1
2
3
```

break and continue Statement

The break and continue statements are used to alter the flow of loops.

The break Statement

The break statement terminates the for loop immediately before it loops through all the items. For example,

```
subjects = ['Physics','Chemestry','Maths','AI','English']
for sub in subjects:
```

```
    if sub == 'AI':
        break
    print(sub)
```

Output

```
Physics
Chemestry
Maths
```

Here, when sub is equal to 'AI', the break statement inside the if condition executes which terminates the loop immediately. This is why AI is not printed.

The continue Statement

The continue statement skips the current iteration of the loop and continues with the next iteration. For example,

```
subjects = ['Physics','Chemestry','Maths','AI','English']
for sub in subjects:
    if sub == 'AI':
        continue
    print(sub)
```

Output

```
Physics
Chemestry
Maths
English
```

Here, when sub is equal to 'AI', the continue statement executes, which skips the remaining code inside the loop for that iteration.

However, the loop continues to the next iteration. This is why English is displayed in the output.

Nested for loops

A loop can also contain another loop inside it. These loops are called nested loops. In a nested loop, the inner loop is executed once for each iteration of the outer loop.

Example: Nested for loops

```
# Print a right-angled triangle pattern of asterisks
rows = 5
for i in range(1,rows+1): # Outer loop for rows
    for j in range(i):    # Inner loop to print '*' in each row
        print("*", end=" ")
    print() # Move to the next line after each row
```

Output

```
*
```

```
* *
* * *
* * * *
* * * * *
```

The outer loop (for i in range(1, rows + 1)) controls how many rows there will be (from 1 to 5).

The inner loop (for j in range(i)) determines how many asterisks are printed in the current row. It will print the number of asterisks. For example, when i is 1, it prints 1 asterisk; when i is 2, it prints 2 asterisks, and so on.

print("*", end=" ") prints an asterisk followed by a space to separate them on the same line.

print() moves the cursor to the next line after all asterisks for a given row have been printed.

Example: Write a function to calculate the factorial of a number. The factorial of a non-negative integer n is the product of all positive integers less than or equal to n. For example, if n is **5**, the return value should be **120** because 1*2*3*4*5 is **120**.

2.3 Python while Loop

In Python, a while loop is used to repeat a block of code until a certain condition is true. For example,

```
number = 1
while number <= 3:
    print(number)
    number = number + 1

Output
1
2
3
```

In the above example, a while loop is used to print the numbers from **1** to **3**. The loop runs as long as the condition number <= 3 is True.

while Loop Syntax

```
while condition:
    # body of while loop
```

Here,

1. The while loop evaluates condition, which is a boolean expression.
2. If the condition is True, the body of the while loop is executed. The condition is evaluated again.
3. This process continues until the condition is False.
4. Once the condition evaluates to False, the loop terminates.

Tip: Keep on updating the variables used in condition inside the loop so that it eventually evaluates to False. Otherwise, the loop keeps running, creating an infinite

loop.

Flowchart of Python while Loop

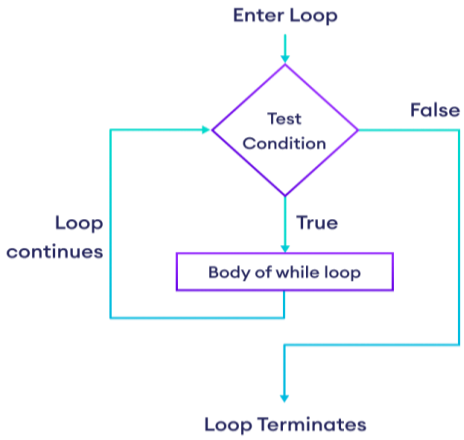


Fig. 2.5 Flowchart of Python while Loop

Example: Python while Loop

```
# Example of Python while Loop
# Print numbers until the user enters 0
number = int(input('Enter a number: '))
# iterate until the user enters 0
while number != 0:
    print(f'You entered {number}.')
    number = int(input('Enter a number: '))
print('The end.')
```

Output

```
Enter a number: 4
You entered 4.
Enter a number: -5
You entered -5.
Enter a number: 100
You entered 100.
Enter a number: 0
The end.
```

Here is how the above program works:

1. It asks the user to enter a number.
2. If the user enters a number other than **0**, it is printed.
3. If the user enters **0**, the loop terminates.

Infinite while Loop

If the condition of a while loop always evaluates to True, the loop runs continuously, forming an infinite while loop. For example,

```
age = 20
```

```
# The test condition is always True
while age > 18:
    print('You can vote')
```

Output

```
You can vote
You can vote
You can vote
.
.
.
```

More on Python while Loop

Python while loop with break statement

We can use a break statement inside a while loop to terminate the loop immediately without checking the test condition. For example,

```
# Example of Python while Loop
while True:
    user_input = input('Enter your name: ')
    # terminate the loop when user enters end
    if user_input == 'end':
        print(f'The loop is ended')
        break
    print(f'Hi {user_input}')
```

Output

```
Enter your name: Diya
Hi Diya
Enter your name: Deepak
Hi Deepak
Enter your name: end
The loop is ended
```

Here, the condition of the while loop is always True. However, if the user enters the end, the loop terminates because of the break statement.

Python while loop with an else clause

A while loop can have an optional else clause - that is executed once the loop condition is False. For example,

```
# Python while loop with an else clause
counter = 0
while counter  <  3:
    print('counter=', counter,'Inside while loop')
    counter = counter + 1
else:
```

```
    print('counter=', counter,'Out to while loop, enter in else block')

Output
counter= 0 Inside while loop
counter= 1 Inside while loop
counter= 2 Inside while loop
counter= 3 Out to while loop, enter in else block
```

Here, on the fourth iteration, the counter becomes **3** which terminates the loop. It then executes the else block and prints Out to while loop, enter in else block.

Note: The else block will not execute if the while loop is terminated by a break statement.

Python for loop vs while loop

The for loop is usually used in the sequence when the number of iterations is known. For example,

```
# Python for loop vs while loop
# loop is iterated 4 times
for i in range(4):
    print(i)

Output
0
1
2
3
```

The while loop is usually used when the number of iterations is unknown. For example,

```
# Python for loop vs while loop
while True:
    name = input("Enter Your Name: ")
    # terminate the loop when user enters exit
    if name == 'Deepak':
        print(f'This is it correct')
        break
    print(f'You Entered Wrong Name:',name)

Output
Enter Your Name: Dipak
You Entered Wrong Name: Dipak
Enter Your Name: deepak
You Entered Wrong Name: deepak
Enter Your Name: Deepak
This is it correct
```

2.4 Python break and continue

In programming, the break and continue statements are used to alter the flow of loops:

1. break exits the loop entirely
2. continue skips the current iteration and proceeds to the next one

Python break Statement

The break statement terminates the loop immediately when it's encountered.

Syntax

Break

Working of Python break Statement

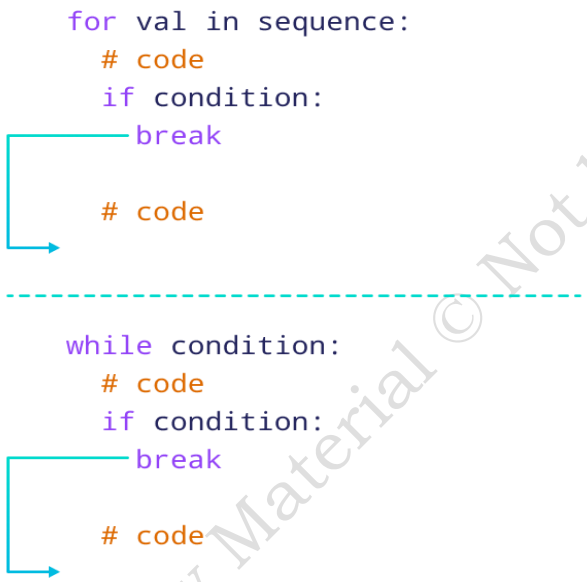


Fig. 2.6 Working of break Statement in Python

The above image shows the working of break statements in for and while loops.

Note: The break statement is usually used inside decision-making statements such as if...else.

Example: break Statement with for Loop

We can use the break statement with the for loop to terminate the loop when a certain condition is met. For example,

```
# Example: break Statement with for Loop
for i in range(5):
    if i == 3:
        break
    print(i)
```

Output

```
0
1
2
```

In the above example, the break statement terminates the loop when i is equal to **3.** Hence, the output doesn't include values after **2**.

Note: We can also terminate the while loop using a break statement.

break Statement with while Loop

We can also terminate the while loop using the break statement. For example,

```
# break Statement with while Loop
i = 0
while i < 5:
    if i == 3:
        break
    print(i)
    i += 1
```

Output

```
0
1
2
```

In the above example, terminate the loop when i is equal to **3**.

Python continue Statement

The continue statement skips the current iteration of the loop and the control flow of the program goes to the next iteration.

Syntax

Continue Working of continue Statement in Python

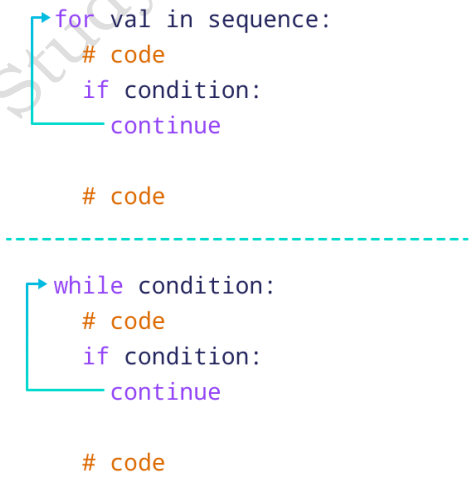


Fig. 2.7 Working of continue Statement in Python

Example: continue Statement with for Loop

We can use the continue statement with the for loop to skip the current iteration of the loop and jump to the next iteration. For example,

```
# Example: continue Statement with for Loop
for i in range(5):
```

```
    if i == 3:
        continue
    print(i)
```

Output

```
0
1
2
4
```

In the above example, the continue statement skips the current iteration when i is equal to **3**, and continues the next iteration. Hence, the output has all the values except **3**.

We can also use the continue statement with a while loop.

continue Statement with while Loop

We can skip the current iteration of the while loop using the continue statement. For example,

```
# Example: continue Statement with while Loop
# Program to print odd numbers from 1 to 10
num = 0
while num < 10:
    num += 1
    if (num % 2) == 0:
        continue
    print(num)
```

Output

```
1
3
5
7
9
```

In the above example, we have used the while loop to print the odd numbers between **1** and **10**. Here,

if (num % 2) == 0:

continue skips the current iteration when the number is even and starts the next iteration.

2.5 Python pass Statement

Python requires an indented block after a colon in if, elif, and else statements. If you have no code to write, pass fulfills this requirement, preventing an IndentationError.

The pass statement is a null statement which can be used as a placeholder within conditional statements (like if, elif, or else blocks). It is used when a block of code is syntactically required but no action is desired for future code.

The syntax of the pass statement is:

pass

Using pass With Conditional Statement

```
# Using pass with conditional statement
num = int(input("Enter a number:"))
if num == 0:
    pass  # No action needed if x is greater than 20
elif num > 0:
    print(f"The number {num} is positive number")
else:
    print(f"The number {num} is negative number")
```

Output

```
Enter a number:10
The number 10 is positive number
Enter a number:-5
The number -5 is negative number
Enter a number:0
```

Here, notice that we have used the pass statement inside the if statement.

However, nothing happens when the pass is executed. It results in no operation (NOP). If you don't use pass in the first indented block of if statement, it will generate error message: IndentationError: expected an indented block.

Note: The difference between a comment and a pass statement in Python is that while the interpreter ignores a comment entirely, pass is not ignored.

Assignment

1. Write a program to print the first 10 even numbers using a while loop.
2. Write a program to find the sum of the digits of a number accepted from the user.
3. Write a program to reverse the number accepted from the user using a while loop.
4. Write a program to check if the input number is Palindrome or not.
5. Write a program to check if the input number is Armstrong or not.

Structural pattern matching in Python

Structural Pattern Matching is a powerful control flow feature introduced in Python 3.10. It allows us to compare a value or object (the "subject") against various patterns and execute code based on the matching pattern. It is a much more advanced and flexible version of a switch-case statement found in other languages.

The basic syntax of structural pattern matching is as follows:

```
match subject:
    case pattern1:
        # Code to execute if subject matches pattern1
    case pattern2:
```

```
        # Code to execute if subject matches pattern2
    # ...
    case _:  # Optional: Wildcard pattern (default case)
        # Code to execute if no other pattern matches
```

```
# Example on Structural Pattern Matching
num = int(input("Enter a number:"))
match num:
    case 1:
        print(f"{num} in words is :","One")
    case 2:
        print(f"{num} in words is :","Two")
    case 3:
        print(f"{num} in words is :","Three")
    case 4:
        print(f"{num} in words is :","Four")
    case 5:
        print(f"{num} in words is :","Five")
    case 6:
        print(f"{num} in words is :","Six")
    case 7:
        print(f"{num} in words is :","Seven")
    case 8:
        print(f"{num} in words is :","Eight")
    case 9:
        print(f"{num} in words is :","Nine")
    case 0: # Special case for 0
        print(f"{num} in words is :","Zero")
    case _:  # Default case for any other number
        print(f"{num} in words is :","Number out of range")
```

Output

```
Enter a number:1
1 in words is : One
Enter a number:2
2 in words is : Two
Enter a number:3
3 in words is : Three
Enter a number:4
4 in words is : Four
Enter a number:5
```

```
5 in words is : Five
Enter a number:6
6 in words is : Six
Enter a number:7
7 in words is : Seven
Enter a number:8
8 in words is : Eight
Enter a number:9
9 in words is : Nine
Enter a number:0
0 in words is : Zero
Enter a number:11
11 in words is : Number out of range
Enter a number:-9
-9 in words is : Number out of range
```

Summary

Control flow in Python refers to the order in which statements are executed in a program. It helps the program make decisions and repeat actions based on given conditions.

The main control flow statements in Python are:

- Conditional statements (if, if-else, elif) – used to make decisions.
- Looping statements (for loop, while loop) – used to repeat a block of code multiple times.
- Control statements (break, continue, pass) – used to control the execution of loops.

Control flow makes programs logical, flexible, and capable of solving real-world problems efficiently.

Check your progress

A. Multiple Choice Questions

1. What is the main purpose of the if statement in Python? (a) To repeat a block of code multiple times (b) To define a new function (c) To execute a block of code only if a specific condition is met (d) To declare a variable
2. Which of the following is the correct syntax for an if statement in Python? (a) if condition { print("Hello") } (b) if condition: print("Hello") (c) if (condition): print("Hello") (d) if condition then print("Hello")
3. What is the purpose of the elif statement? (a) To specify a block of code to be executed if the initial if condition is false and there are no further conditions to check (b) To add an alternative condition to be checked if the previous if or elif conditions are false (c) To terminate the program if a condition is not met (d) To define a new type of loop

4. What will be the output of following code?: (a) A (b) B (c) C (d) No output

```
x = 10
if x > 15:
        print("A")
elif x == 10:
        print("B")
else:
        print("C")
```

5. What will be the output of this Python code? (a) TRUE (b) FALSE (c) FALSE DONE (d) TRUE DONE

```
if 4 + 5 == 10:
        print("TRUE")
else:
        print("FALSE")
print("DONE")
```

6. How do you indicate a block of code under an if statement in Python? (a) Using curly braces {} (b) Using parentheses () (c) Using indentation (d) Using semicolons;
7. What is the outcome if the condition in an if statement evaluates to False, and there is no elif or else block? (a) A syntax error will occur (b) The program will stop (c) The program will continue to the next statement after the if block, essentially doing nothing if the condition is false (d) A runtime error will occur.
8. Which operator is used to check for equality in an if statement condition? (a) = (b) == (c) != (d) is
9. Can an if statement be nested inside another if statement in Python? (a) No, nested if statements are not allowed in Python (b) Yes, to handle more complex decision-making scenarios (c) Yes, but only up to one level of nesting (d) Only within functions
10. What will be the output of the following code? (a) Small (b) Medium (c) Large (d) No output

```
a = 5
if a < 3:
        print("Small")
elif a < 7:
        print("Medium")
else:
        print("Large")
```

11. What is the output of the following Python code? (a) ['ab', 'cd'] (b) ['AB', 'CD'] (c) [None, None] (d) None of the mentioned.

```
x = ['ab', 'cd']
```

```
for i in x:
        i.upper()
print(x)
```

12. Which type of loop is guaranteed to execute at least once in Python? (a) for loop (b) while loop (c) do-while loop (d) Neither a nor b.
13. What is the primary difference between a for loop and a while loop in Python? (a) for loops are for definite iteration, while while loops are for indefinite iteration (b) for loops are always faster than while loops (c) while loops cannot iterate over sequences like lists or strings (d) for loops cannot use the break or continue statements
14. When does the else block associated with a for loop execute? (a) It is executed only if a break statement is encountered in the loop (b) It is executed only if the loop completes without encountering a break statement (c) It is always executed after the loop completes (d) It is a syntax error to use else with a for loop.
15. What will be the output of the following code? (a) 2 4 6 8 10 ..... (b) 2 4 (c) 2 3 (d) Error

```
i = 2
while True:
        if i % 3 == 0:
                break
        print(i, end=" ")
        i += 2
```

B. Fill in the Blanks

1. Control structures in Python dictate the _____ of execution of a program.
2. The three main types of control structures are sequential, selection (or decision-making), and _____.
3. Statements that execute one after another in a linear fashion represent _____ control flow.
4. Decision-making control structures use _____ statements to execute code selectively based on whether conditions evaluate to True or False.
5. The _____ statement is the simplest conditional statement, executing a block of code only if its condition is true.
6. The _____ statement allows for an alternative action to be performed if the if condition evaluates to false.
7. To check multiple conditions in sequence, you would typically use an _____ ladder.
8. Repetition control structures, also known as _____, allow a program to execute a block of code multiple times.
9. A _____ loop is used to iterate over a sequence (like a list, tuple, string).
10. A _____ loop repeats a block of code as long as a specified condition is true.

11. The _____ statement is used to exit a loop prematurely.
12. The _____ statement skips the current iteration of a loop and moves on to the next.
13. The _____ statement is a null operation; it does nothing and serves as a placeholder.
14. The code within an if, elif, else, for, or while block must be properly _____ in Python.
15. Using break, continue, and pass statements within loops and conditionals helps manage and control the _____ of the program effectively.

C. State whether True or False

1. Control structures determine the order in which statements are executed in a Python program.
2. Python primarily uses curly braces {} to define code blocks for if statements and loops.
3. The else block associated with an if statement is executed if the if condition is True.
4. The elif statement allows you to check multiple conditions sequentially if the preceding if or elif conditions are False.
5. A for loop is primarily used for indefinite iteration, where the number of repetitions is unknown.
6. A while loop continues to execute as long as its condition remains True.
7. The break statement exits only the innermost loop in a set of nested loops.
8. The continue statement terminates the entire loop and moves control to the statement immediately following the loop.
9. The pass statement in Python is used to indicate an empty code block.
10. The else block associated with a for or while loop executes only if the loop completes *without* encountering a break statement.

Session 3. Python Data Types

3.1 Python Numbers

In Python, numbers are a fundamental data type used to store numerical values and perform mathematical operations. There are three main numeric types in Python: int, float, and complex.

Integer (int): These represent whole numbers, positive or negative of unlimited length, without a fractional part. Examples include 5, -12, and 0. You can even use underscores for readability in large numbers, like 1_000_000,

Float (float): These represent real numbers with a decimal point or in scientific notation (e.g., using 'e' or 'E'). Floats are crucial for calculations requiring precision, like those in

scientific or financial tasks. Float values can be positive or negative. Examples include 5.5, -12.07, and 0.0.

Complex (complex): These numbers have both a real and an imaginary part, expressed as a + bj, where a is the real part and b is the imaginary part, with j representing the imaginary unit. They're useful in fields like computer graphics and scientific computing.

We can use the type() function to know the type of data.

The following example illustrates the type of data values.

```
# Show the data type of the variables
num1 = 10
print(num1, 'is of type', type(num1))

num2 = 99.99
print(num2, 'is of type', type(num2))

num3 = 5+2j
print(num3, 'is of type', type(num3))
```

Output

```
10 is of type <class 'int'>
99.99 is of type <class 'float'>
(5+2j) is of type <class 'complex'>
```

3.2 Python List

In Python, lists allow us to store multiple items in a single variable. For example, if you need to store the ages of all the students in a class, you can do this task using a list.

Lists are similar to arrays (dynamic arrays that allow us to store items of different data types) in other programming languages.

3.2.1 Create a Python List

We create a list by placing elements inside square brackets [], separated by commas. For example,

```
# a list of three elements
ages = [17, 19, 18]
print(ages)
```

Output

```
[17, 19, 18]
```

Here, the ages list has three items.

3.2.2 List Items of Different Types

Python lists are very flexible. We can also store data of different data types in a list. For example,

```
# a list containing strings, numbers and another
```

```
list
student = ['Jayant', 18, 'AI', [2, 4]]
print(student)

Output
['Jayant', 18, 'AI', [2, 4]]
```

It is also possible to create an empty list. The output will be square bracket [].

```
# an empty list
empty_list = []
print(empty_list)

Output
[]
```

3.2.3 List Characteristics

Python lists are one of the most flexible and widely used built-in data types. They are highly versatile and offer several key features:

1. **Ordered:** Lists maintain the order of its elements. The list items are stored and retrieved in the sequence they were added. Each element has a specific index, with the first element at index 0.
2. **Mutable:** List items can be modified after creation. You can add, remove, or change elements within a list, notes Educative.
3. **Dynamic:** Lists can grow or shrink in size as needed, adapting to the number of elements.
4. **Heterogeneous:** A single list can contain elements of different data types, including integers, floats, strings, objects, and even other lists.
5. **Indexed and Sliceable:** Elements can be accessed, updated, or removed using their index position. Slicing.
6. **Iterable:** Lists can be iterated over using loops (like for loops) or list comprehensions.
7. **Allows Duplicates:** Lists can have duplicate elements, and each instance is treated independently.
8. **Supports Nesting:** Lists can contain other lists as elements.

3.2.4 Accessing list elements in Python

You can access individual elements in a Python list using different ways. Indexing, and slicing are the two common methods of accessing the list elements.

1. Indexing (for single elements)

a) **Positive indexing:** Python lists use zero-based indexing. The first element has an index of 0, the second element has an index of 1, and so on.

b) **Negative indexing:** You can also use negative indexing to access elements from the end of the list. In this case, -1 refers to the last element, -2 refers to the second-to-last element, and so on

Example … will illustrate to access a list element using indexing.

```
# Accessing list elements using indexing
subjects = ['English','Physics','Chemistry','Maths','AI']
# access the first item
print('First subject is : ', subjects[0])
# access the third item
print('Third subject is : ', subjects[2])
# access the last item
print('Last subject is  : ', subjects[-1])
```

Output

```
First subject is :  English
Third subject is :  Chemistry
Last subject is  :  AI
```

2. Slicing (for multiple elements)

List slicing allows you to extract a sublist or a range of elements from the list by specifying a start and end index, and an optional step size.

The syntax is: my_list[start:end:step]

a) **start:** The starting point of the list (inclusive) to begin the slice. If omitted, it defaults to 0.

b) **end:** The index where the slice ends (exclusive). If omitted, it defaults to the end of the list.

c) **step:** The step size for selecting elements (defaults to 1). A negative step reverses the order of the slice.

Example … will illustrate to access elements of the list using slicing.

```
# Accessing list elements using slicing

my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
print("The original list is : ", my_list)

# list elements from index 0 to 3 (index 3 is not included)
print("list elements from index 0 to 3 :", my_list[0:3])

# list elements from index 3 to index 6 (index 7 is not included)
print("list elements from index 3 to index 6 :", my_list[3:7])

# list elements from index 1 to index -4 (index -3 is not included)
print("list elements from index 1 to index -4:", my_list[1:-3])
```

Output

```
The original list is :  ['p', 'r', 'o', 'g', 'r', 'a', 'm']
list elements from index 0 to 3: ['p', 'r', 'o']
list elements from index 3 to index 6:['g', 'r', 'a', 'm']
list elements from index 1 to index -4:['r', 'o', 'g']
```

3. Omitting Start and End Indices in Slicing

If you omit the start index, the slicing starts from the first element. Similarly, if you omit the last index, the slicing ends at the last element. For example,

```
# Omitting Start and End Indices in Slicing

my_list = ['p','r','o','g','r','a','m']
print("The original list is: ", my_list)

# list with elements from index 3 to last
print("list with elements from index 3 to last:", my_list[3: ])

# list with elements from first element to index -5
print("list with elements from first element to index -5:", my_list[:-4])

# omitting both start and end index
# get a list from start to end items

print("list with elements from start to end:", my_list[:])
```

Output

```
The original list is:  ['p', 'r', 'o', 'g', 'r', 'a', 'm']
list with elements from index 3 to last: ['g', 'r', 'a', 'm']
list with elements from first element to index -5: ['p', 'r', 'o']
list with elements from start to end: ['p', 'r', 'o', 'g', 'r', 'a', 'm']
```

Note: If the specified index does not exist in a list, Python throws the IndexError exception.

3.2.5 Adding Elements to a Python List

Lists are mutable and it is possible to change items of a list. To add an item to the end of a list, use append() method. Example … illustrate this ,

```
# Adding Elements to a Python List
fruits = ['apple', 'banana', 'orange']
print('Original List:', fruits)
fruits.append('cherry')
print('Updated List:', fruits)
```

Output

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'orange', 'cherry']
```

3.2.6 Adding Elements at the Specified Index

It is possible to insert an element at the specified index to a list using the insert() method. Example... illustrate this,

```
# Adding Elements at the Specified Index
fruits = ['apple', 'banana', 'orange']
print("Original List:", fruits)
fruits.insert(2, 'cherry')
print("Updated List:", fruits)
```

Output

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'cherry', 'orange']
```

3.2.7 Add Elements to a List From Other Iterables

The list extend() method adds all the items of the specified iterable, such as list, tuple, dictionary or string , to the end of a list. For example,

```
# Adding Elements to List From Other Iterables
numbers = [1, 3, 5]
print('Numbers:', numbers)
even_numbers = [2, 4, 6]
print('Even numbers:', numbers)
# adding elements of one list to another
numbers.extend(even_numbers)
print('Updated Numbers:', numbers)
```

Output

```
Numbers: [1, 3, 5]
Even numbers: [1, 3, 5]
Updated Numbers: [1, 3, 5, 2, 4, 6]
```

3.2.8 Changing List Items

It is also possible to change the items of a list by assigning new values using the = operator. For example,

```
# Changing List Items
colors = ['Red', 'Black', 'Green']
print('Original List:', colors)
# change the first item to 'Blue'
colors[0] = 'Blue'
# change the third item to 'Purple'
colors[2] = 'Purple'
print('Updated List:', colors)
```

Output

```
Original List: ['Red', 'Black', 'Green']
```

```
Updated List: ['Blue', 'Black', 'Purple']
```

3.2.9 Removing an Item From a List

It is possible to remove the specified item from a list using the remove() method. For example,

```
# Removing an Item From a List
numbers = [11,20,22,33,44,55]
print("Original List :",numbers)
# remove third item 4 from the list
numbers.remove(20)
print("List after removing 20 :",numbers)
```

Output

```
Original List : [11, 20, 22, 33, 44, 55]
List after removing 20 : [11, 22, 33, 44, 55]
```

3.2.10 Remove One or More Elements of a List

Instead of using the remove() method, you can delete an item from a list using the del statement. The del statement can also be used to delete multiple elements or even the entire list.

```
# Removing One or More Elements of a List
names = ['Anil','Sunil','Kareem','Lila','Jayant','Kanak']
# delete the item at index 1
del names[1]
print("The original list : ",names)
# delete items from index 1 to index 3
del names[1:3]
print("List after removing items at index 1 to 3 :",names)
# delete the entire list
del names
# Error! List doesn't exist.
print(names)
```

Output

```
The original list :  ['Anil', 'Kareem', 'Lila', 'Jayant', 'Kanak']
List after removing items at index 1 to 3 : ['Anil', 'Jayant', 'Kanak']
NameError: name 'names' is not defined
```

3.2.11 Finding Python List Length

It is possible to find the number of elements in the list. To find the number of elements (length) of a list, use the built-in len() function. For example,

```
# Finding Python List Length
names = ['Anil','Sunil','Kareem','Lila','Jayant','Kanak']
print("The original list : ",names)
print('Total names in list:',len(names))
```

```
The original list : ['Anil', 'Sunil', 'Kareem', 'Lila', 'Jayant', 'Kanak']
Total names in list: 6
```

3.2.12 Iterating Through a List

We can use a for loop to iterate over the elements of a list. For example,

```
# Iterating Through a List
fruits = ['apple','banana','cherry','orange']
# iterate through the list
for fruit in fruits:
    print(fruit)
apple
banana
cherry
orange
```

Python List Methods

Python has many useful list methods that make it really easy to work with lists.

Method	Description
append()	Adds an item to the end of the list
extend()	Adds items of lists and other iterables to the end of the list
insert()	Inserts an item at the specified index
remove()	Removes the specified value from the list
pop()	Returns and removes item present at the given index
clear()	Removes all items from the list
index()	Returns the index of the first matched item
count()	Returns the count of the specified item in the list
sort()	Sorts the list in ascending/descending order
reverse()	Reverses the item of the list
copy()	Returns the shallow copy of the list

3.4 Python Tuple

A tuple is a collection similar to a Python list. The primary difference is that we cannot modify a tuple once it is created.

In Python, a tuple is an ordered collection of elements, similar to a list. However, the key difference is that **tuples are immutable**, meaning you cannot change, add, or remove items after a tuple has been created.

3.4.1 Characteristics of Tuple

This immutability gives tuples several distinct characteristics.

1. **Ordered:** Elements in a tuple maintain their defined order, and allow to access by index.

2. **Immutable:** Once a tuple is created, its elements cannot be modified. Attempting to modify a tuple will result in a TypeError.
3. **Allows Duplicates:** Tuples can contain duplicate values, as the items are accessed by index.
4. **Heterogeneous:** A single tuple can hold elements of different data types, including integers, strings, floats, and even other tuples.
5. **Indexed:** Each item in a tuple has an index, starting from 0 for the first element. Negative indexing also works, with -1 referring to the last item, -2 to the second last, and so on.
6. **Nestible:** Tuples can contain other tuples to create nested structures.
7. **Iterable:** Tuples support iteration, allowing to traverse their elements using loops or comprehensions.
8. **Sliceable:** You can extract portions of a tuple using slicing, which returns a new tuple

3.4.2 Creating a Python Tuple

Tuples are created in two ways:

Creating a tuple using parentheses: The most common way is to enclose the elements in parentheses () and separate them with commas. You can also create a tuple by simply separating elements with commas.

```
# Creating a Tuple using parentheses
my_tuple = (1, 2.5, 3, -5, 9)
print("Tuple created with parentheses :",my_tuple)

# Creating a Tuple without parentheses (tuple packing)
my_tuple = 10, 22.5, 26.5, 24.5
print("Tuple created without parentheses :",my_tuple)
```

Output

```
Tuple created with parentheses : (1, 2.5, 3, -5, 9)
Tuple created without parentheses : (10, 22.5, 26.5)
```

Creating a tuple using the tuple() constructor: You can convert other iterable objects like lists or strings into tuples using the built-in tuple() function.

```
# Creating a tuple using the tuple() constructor
tuple_constructor = tuple(('June', 'July', 'August'))
print("Tuple created using constructor:", tuple_constructor)
```

Output

```
Tuple created using constructor: ('June', 'July', 'August')
```

Different Types of Python Tuples

We can create the different types of tuples in python such as Empty Tuple, Single item Tuple, Tuple of different data types, Tuple of mixed data types, as illustrated below.

```
# Creating different types of tuple
# Empty tuple
empty_tuple = ()
print("Empty tuple created :",empty_tuple)

# tuple with a single item, requires a trailing comma
single_element_tuple = (49,)
print("tuple created with a single item :",single_element_tuple)

# tuple created from a list
list_data = [1.1, 2.2, 3.3, 4.4, 5.5]
tuple_from_list = tuple(list_data)
print("tuple created from a list :",tuple_from_list)

# tuple of string types
names = ("Anil", "Sunil", "Kareem")
print ("tuple of string types :",names)

# tuple of float types
temperature = (31.5, 33.4, 34.2)
print("tuple of float types :",temparature)

# tuple of mixed data types string and integer
mixed_tuple = (2, 'Hello', 'Students')
print("tuple of string and integer :",mixed_tuple)
```

Output

```
Empty tuple created : ()
tuple created with a single item : (49,)
tuple created from a list : (1.1, 2.2, 3.3, 4.4, 5.5)
tuple of string types : ('Anil', 'Sunil', 'Kareem')
tuple of float types : (31.5, 33.4, 34.2)
tuple of string and integer : (2, 'Hello', 'Students')
```

3.4.3 Accessing Tuple Items

Each item in a tuple is associated with a number, known as an index. The index always starts from **0**, which means the first item of a tuple is at index **0**, the second item is at index **1,** and so on.

```
# Accessing Items Using Index
subjects = ('Physics', 'Chemistry', 'Maths', 'AI')

# access the first item
print("the first subject is :", subjects[0])
```

```
# access the third item
print("the third subject is :", subjects[2])

the first subject is : Physics
the third subject is : Maths
```

3.4.5 Deleting Tuples

We cannot delete individual items of a tuple. However, we can delete the tuple itself using the del statement. For example,

In Python, tuples are immutable, meaning you cannot delete individual elements within a tuple. However, you can delete the entire tuple object from memory using the del keyword.

```
# Deleting a tuple
subjects = ('Physics', 'Chemistry', 'Maths', 'AI')
print("Original tuple :",subjects)
del subjects
print("Accessing tuple after deletion will result in a NameError")
print(subjects)

Output
Original tuple : ('Physics', 'Chemistry', 'Maths', 'AI')
Accessing tuple after deletion will result in a NameError
NameError: name 'subjects' is not defined
```

3.4.6 Modification of Tuple

Python tuples are immutable, meaning once it is created, its elements cannot be directly changed, added, or removed. Attempting to modify a tuple directly will result in a TypeError.

```
# Try to modify a tuple
subjects = ('Physics', 'Chemistry', 'Maths', 'AI')
subjects(o) = 'English'

Output
SyntaxError: cannot assign a function call here. Maybe you meant '==' instead of '='?
```

However, there is a wayout to modify a tuple. The most common approach is to convert the tuple into a list, modify the list as needed (add, change, or remove elements), and then convert the list back into a tuple.

```
# Modification of Tuple
original_tuple = ('English','Physics', 'Chemistry', 'Maths')
print(f"Original tuple: {original_tuple}")

# Convert to a list
mutable_list = list(original_tuple)
```

```
# Modify the list
mutable_list[2] = "Electronics" # Change an element
mutable_list.append("AI")       # Add an element
del mutable_list[0]             # Remove an element

# Convert back to a tuple
modified_tuple = tuple(mutable_list)
print(f"Modified tuple: {modified_tuple}")
```

3.4.7 Finding length of Tuple

We use the len() function to find the number of items present in a tuple. For example,

```
# Finding length of tuple
subjects = ('Physics', 'Chemistry', 'Maths', 'AI')
print('Total Subjects :', len(subjects))

Output
Total Subjects : 4
```

3.4.8 Iterating Through a Tuple

Tuples, like lists, are iterable objects in Python. You can loop through their elements one by one. Tuples are immutable, so while iterating, you can access and use the elements, but you cannot change them within the tuple itself. There are several ways to iterate over a tuple, each suitable for different situations

1. Using a for loop

The most common way to iterate through a tuple is to use a for loop. This method directly accesses each item in the tuple during each iteration

```
# Iterating Through a Tuple using for loop
subjects = ('Physics', 'Chemistry', 'Maths', 'AI')

for subjects in subjects:
    print(subjects)

Output

Physics
Chemistry
Maths
AI
```

2. Using for loop with range() and len()

If you need to access both the element and its index during iteration, you can combine a for loop with range(len(tuple_name))

```
# Iterating Through a Tuple using for loop
```

```
subjects = ('Physics', 'Chemistry', 'Maths', 'AI')

for i in range(len(subjects)):
    print(f"Index: {i}, Item: {subjects[i]}")
```

Output

```
Index: 0, Item: Physics
Index: 1, Item: Chemistry
Index: 2, Item: Maths
Index: 3, Item: AI
```

3. Using a while loop

A while loop can also be used to iterate through a tuple, specially when more control over the iteration steps is needed,

```
# Iterating Through a Tuple using for loop
subjects = ('Physics', 'Chemistry', 'Maths', 'AI')
i=0
while i < len(subjects):
    print(subjects[i])
    i+=1

Physics
Chemistry
Maths
AI
```

3.4.9 Checking an Item Exists in the Tuple

Python provides several ways to check if a particular item exists in a tuple. The most common and efficient way is to use the in. It returns True if the item is present, and False otherwise.

```
# Check if an Item Exists in the Tuple
colors = ('red', 'orange', 'blue')
print("Is black exist in color?", 'black' in colors)
print("Is blue exist in color?", 'blue' in colors)
```

Output

```
Is black exist in color? False
Is blue exist in color? True
```

3.5 Python String

In Python, a string is a sequence of characters. For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

The single quotes or double quotes are used to represent a string in Python. For example,

```
# create a string using double quotes
```

```
string1 = "Python programming"
# create a string using single quotes
string1 = 'Python programming'
```

Here, a string variable named *message* is created and initialized with the string "Python Programming".

Example: Python String

```
# create string type variables
name = "Diya"
print(name)
message = "My name is"
print(message, name)

Output
Diya
My name is Diya
```

3.5.1 Accessing String Characters in Python

In Python, strings are sequences of characters. You can access individual characters within a string using **indexing** and extract portions of a string (sub-strings) using **slicing**. Python strings are immutable, meaning you cannot change individual characters in a string once it's created. Instead, string operations that appear to modify a string actually create a new string based on the original.

1. Indexing

Each character in a string has a numerical position called an index. Python uses zero-based indexing, where:

1. The first character is at index 0.
2. The second character is at index 1, and so on.
3. The last character's index is length - 1

You access characters using square brackets [] and the character's index.

```
# Access String Characters
message = "Hello, World!"

# Access the character at index 0 (the first character)
print("First character at index [0]:",message[0])

# Access the character at index 7 (the 8th character)
print("Character at index [8]:",message[7])

# Access the character at the last index
# Using len() function to get the length - 1
print("Last character is:",message[len(message)-1])
```

Output
First character at index [0]: H
Character at index [8]: W
Last character is: !

Negative Indexing

You can also use negative indices to access characters from the end of the string. Index -1 refers to the last character. Index -2 refers to the second-to-last character, and so on.

```
# Access String Characters using Negative indexing
my_string = "INDIA"
# Access the last character
print("Last character:",my_string[-1])
# Access the second-to-last character
print("Secondlast character:",my_string[-2])
```

Output
Last character: A
Secondlast character: I

2. String slicing

Slicing allows us to extract a substring from a string by specifying a range of indices. The syntax for slicing is string[start:stop:step].

1. start (optional): The starting index of the slice (inclusive). Defaults to 0 if omitted.
2. stop (optional): The ending index of the slice (exclusive). Defaults to the end of the string if omitted.
3. step (optional): The interval between characters (default is 1). A positive value slices from left to right, while a negative value slices from right to left

```
# Access String Characters using Slicing
message = "Hello, World!"

# Extract a substring from index 0 to 4 (exclusive of 5)
print("substring from index 0 to 4:", message[0:5])

# Extract a substring from index 7 to the end
print("substring from index 7 to the end:", message[7:])

# Extract a substring from the beginning to index 5 (exclusive)
print("substring from the beginning to index 5:", message[:5])

# Extract the entire string
print("the entire string:",message[:])

# Extract every other character
print("alternate characters:",message[::2])
```

```
# Reverse the string
print("reversal of string:",message[::-1])
```

Output

```
substring from index 0 to 4: Hello
substring from index 7 to the end: World!
substring from the beginning to index 5: Hello
the entire string: Hello, World!
alternate characters: Hlo ol!
reversal of string: !dlroW ,olleH
```

In Python, strings are immutable. That means the characters of a string cannot be changed. For example,

```
message = 'Python'
message[0] = 'H'
print(message)
```

Output

```
TypeError: 'str' object does not support item assignment
```

However, we can assign the variable name to a new string. For example,

```
# Assign new string to message variable
message = 'Python'
print("Original string:",message)
message = 'Program'
print("String changed to:",message)
```

Output

```
Original string: Python
String changed to: Program
```

Python Multiline String

We can also create a multiline string in Python. For this, we use triple double quotes """ or triple single quotes '''. For example,

```
# multiline string
message = """
Hello Students,
Let us learn Python Programming
It is a programming language
"""
print("Multiline string is:",message)
```

Output

```
Multiline string is:
Hello Students,
```

```
Let us learn Python Programming
It is a programming language
```

Iterating Through a Python String

We can iterate through a string using a for loop. For example,

```
# iterating through string
greet = 'Hello'

for letter in greet:
    print(letter)
Output
H
e
l
l
o
```

Python String Length

In Python, we use the len() method to find the length of a string. For example,

```
# count length of string
greet = 'Hello'
print("String:",greet)
print("Length of string:",len(greet))

Output
String: Hello
Length of string: 5
```

String Membership Test

We can test if a substring exists within a string or not, using the keyword in.

```
# String Membership Test
print('a' in 'program') # True
print('at' not in 'battle') # False

Otput
True
False
```

Methods of Python String

Besides those mentioned above, there are various string methods present in Python. Here are some of those methods:

Methods	Description
upper()	Converts the string to uppercase

lower()	Converts the string to lowercase
partition()	Returns a tuple
replace()	Replaces substring inside
find()	Returns the index of the first occurrence of substring
rstrip()	Removes trailing characters
split()	Splits string from left
startswith()	Checks if string starts with the specified string
isnumeric()	Checks numeric characters
index()	Returns index of substring

Escape Sequences in Python

The escape sequence is used to escape some of the characters present inside a string.

Suppose we need to include both a double quote and a single quote inside a string,

```
>>> message = "He said, "What's there?""
>>> print(message) # throws error
```

Since strings are represented by single or double quotes, the compiler will treat "He said, " as a string. Hence, the above code will cause an error.

To solve this issue, we use the escape character \ in Python.

A list of all the escape sequences supported by Python is given below.

Escape Sequence	Description
\\	Backslash
\'	Single quote
\"	Double quote
\a	ASCII Bell
\b	ASCII Backspace
\f	ASCII Formfeed
\n	ASCII Linefeed
\r	ASCII Carriage Return
\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab
\ooo	Character with octal value ooo
\xHH	Character with hexadecimal value HH

3.6 Python String Operations

Python offers a rich set of built-in operations and methods for manipulating strings. Strings are fundamental for handling text data in Python applications, ranging from simple text processing to complex natural language processing.

Many operations can be performed with strings, which makes it one of the most used data types in Python. A comprehensive list is given below and few of them explained.

1. Basic operations

Common string operations include:

1. **Concatenation (+):** Joins strings.
2. **Repetition (*):** Repeat a string.
3. **Length (len()):** Returns the number of characters.
4. **Indexing:** Accesses individual characters using zero-based or negative indexing.
5. **Slicing:** Extracts substrings using a range of indices.
6. **Checking for Substring (in):** Determines if a substring is present.

2. String methods

Python provides various built-in string methods for tasks such as:

1. **Case Conversion:** upper(), lower(), title(), capitalize().
2. **Trimming:** strip(), lstrip(), rstrip() to remove whitespace.
3. **Searching:** find() and index() to locate substrings.
4. **Replacing:** replace() to substitute substrings.
5. **Splitting and Joining:** split() to divide a string into a list and join() to combine list items into a string.

3. String formatting

Ways to format strings include:

1. **f-strings:** Embedding expressions directly in strings (Python 3.6+).
2. **format() method:** Using placeholders for values.
3. **Modulo operator (%)**: An older method for formatting with placeholders.

3.6.1 Concatenating strings in Python

String concatenation is the process of joining two or more strings together to create a new, single string. There are several methods to do this. The most common method is to use the + operator as illustrated below.

```
# Concatenating strings using + operator
str1 = "Hello"
str2 = "World"
# Joining two strings
str_new = str1 + str2
print("After joining two strings: ",str_new)
# Joining with space between two strings
str_new = str1 + " " + str2
print("After joinging strings with space:",str_new)

Output
After joining two strings:  HelloWorld
After joining strings with space: Hello World
```

2. Using the join() method

Using **+** operator is the less efficient method for concatenating many strings repeatedly in loops, because each + operation creates a new string object in memory. The join() method is the most efficient way to concatenate a sequence of strings (like a list or tuple), when dealing with a large number of strings or within loops. It minimizes the creation of intermediate strings.

```
# Concatenating strings using join () method
# Joining strings with space
words = ["What", "is", "your", "name"]
sentence = " ".join(words)
print("Sentence by joining words:",sentence)
# Join with a different separator
wording = ",".join(words)
print("Words of sentence separated by comma:",wording)
```

Output

```
Sentence by joining words: What is your name
Words of sentence separated by comma: What,is,your,name
```

3. String Formatting using f-Strings

f-strings provide a concise and readable way to embed expressions directly into string literals, making them excellent for string concatenation and formatting.

```
# String Formatting using f-Strings
name = "Ajay"
age = 11
message = f"My name is {name} and I am in Grade {age}."
print(message)
```

Output

```
My name is Ajay and I am in Grade 11.
```

4. Repetition using the * operator

The * operator can be used to repeat a string a specified number of times, effectively concatenating it to itself.

```
# Repetition of string using the * operator
repeated_string = "Hello! " * 3
print(repeated_string) # Output: Hello! Hello! Hello!
```

Output

```
Hello! Hello! Hello!
```

These are the few methods for string concatenation. You can select the most appropriate method depending on the task.

3.6.2 Compare Two Strings

String comparison is a fundamental operation in Python. It determines the relationship between two strings. This is used in sorting, searching, filtering data, and user input validation.

There are different ways to compare two strings.

1. Using equality and relational operators

Python offers built-in comparison operators for use with strings:

1. ==: Checks for equality. Returns True if both strings are identical (including case), and False otherwise.
2. !=: Checks for inequality. Returns True if the strings differ, and False otherwise.
3. <, <=, >, >=: Perform comparison according to alphabetical order.

```
# Comparing two strings
string1 = "Hello"
string2 = "Hello"
string3 = "World"
print(string1 == string2)  # Output: True
print(string1 != string3)  # Output: True
print(string1 < string3)   # Output: True (H comes before W)

Output
True
True
True
```

2. Case-insensitive comparison

By default, string comparisons in Python are case-sensitive. To compare strings without regard to case, convert both strings to a common case (lowercase or uppercase) before comparison. It's generally recommended to use == for comparing string values, and is only when checking for object identity

```
# Comparing two case sensitive strings
string1 = "Hello"
string2 = "hello"
# Same strings of different case are not equal
print(string1 == string2)                  # Output: False
# Using .lower() for case-insensitive comparison
print(string1.lower() == string2.lower())   # Output: True
# Using .upper() for case-insensitive comparison
print(string1.upper() == string2.upper())   # Output: True

Output
False
True
True
```

3. Comparing string identity

The is and is not operators check if two variables refer to the identical object in memory, not just if their values are equal.

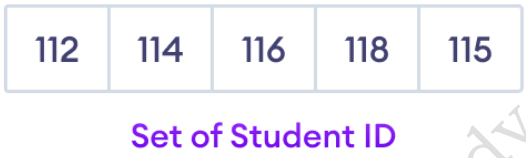
```
# Comparing string identity
string1 = "Hello"
string2 = "Hello"
string3 = "hello"
print(string1 is string2)  # Output: True
print(string1 is string3)  # Output: False

Output
True
False
```

These are some fundamental ways to compare strings in Python. There are other methods also. The appropriate method depends on the specific requirements of the comparison task.

3.7 Python Sets

A set in Python is an unordered collection of unique elements. The elements within a set cannot be duplicated. Suppose you want to store information about student IDs, a set is suitable since student IDs cannot have duplicates.



3.7.1 Creating a Set in Python

You can create sets in Python using two ways.

1. Using curly braces {}

This is the most common way to create a set, where all the set elements are placed inside curly braces {}, separated by commas. A set can have any number of items and they may be of different types (integer, float, tuple, string, etc.). ISet cannot have mutable elements like lists, sets or dictionaries as its elements.

Let's see an example,

```
# creating a set using curly braces {}
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)
# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)
# create a set of mixed data types
mixed_set = {'Mixed', 'Fruit','Juice', '@', 'Rs.', 50}
```

```
print('Set of mixed data types:', mixed_set)
```

Output

```
Student ID: {112, 114, 115, 116, 118}
Vowel Letters: {'e', 'u', 'o', 'i', 'a'}
Set of mixed data types: {'Fruit', 50, '@', 'Mixed', 'Rs.', 'Juice'}
```

In the above example, we have created different types of sets by placing all the elements inside the curly braces {}.

Note: When you run this code, you might get output in a different order. This is because the set has no particular order.

Duplicate Items in a Set

If you include duplicate values, they are automatically removed, as sets store only unique elements.

```
# Duplicate items automatically removed in set
numbers = {2, 4, 6, 6, 2, 8}
print(numbers)
```

Output

```
{8, 2, 4, 6}
```

2. Using the set() constructor

You can create a set from any iterable (like a list, tuple, or string) using the built-in set() function.

```
# creating a set using set() constructor
numbers = set([1, 2, 2, 3, 4])
print(numbers) # Output: {1, 2, 3, 4}
```

Output

```
{1, 2, 3, 4}
```

In the output, you can see there are no duplicate items in the set as a set cannot contain duplicates.

Creating an Empty Set

To create an empty set, you must use the set() constructor without any arguments. Using empty curly braces {} will create an empty dictionary.

```
# create an empty set
empty_set = set()
# create an empty dictionary
empty_dictionary = {}
# check data type of empty_set
print('Data type of empty_set:', type(empty_set))
# check data type of dictionary_set
```

```
print('Data type of empty_dictionary:', type(empty_dictionary))

Output
Data type of empty_set: <class 'set'>
Data type of empty_dictionary: <class 'dict'>
```

Here,

a) *empty_set* – an empty set created using set()
b) *empty_dictionary* – an empty dictionary created using {}
c) type() function is used to know the class of *empty_set* and *empty_dictionary*.

Adding and Updating Set Items in Python

Sets are mutable. However, since they are unordered, indexing has no meaning. We cannot access or change an element of a set using indexing or slicing. The set data type does not support it.

Adding Items to a Set in Python

In Python, we use the add() method to add an item to a set. For example,

```
# Adding element in set
numbers = {25, 34, 44, 22}
print('Initial Set:',numbers)
# using add() method
numbers.add(55)
print('Set after adding new element:', numbers)

Output
Initial Set: {25, 34, 44, 22}
Set after adding new element: {34, 44, 22, 55, 25}
```

In the above example, a set named *numbers* is created. The function add() adds **32** to our set.

Updating Python Set

The update() method is used to update the set with items other collection types (lists, tuples, sets, etc). For example,

```
# Updating set
subjects = {'Physics', 'Chemistry', 'Maths'}
print("Academic subjects:",subjects)
voc_subjects = ['AI', 'english']
print("Vocational subjects:",voc_subjects)
# using update() method
subjects.update(voc_subjects)
print("All subjects:",subjects)
Output
Academic subjects: {'Maths', 'Chemistry', 'Physics'}
```

```
Vocational subjects: ['AI', 'english']
All subjects: {'Physics', 'Chemistry', 'AI', 'Maths', 'english'}
```

Remove an Element from a Set

We use the discard() method to remove the specified element from a set. For example,

```
# Removing element from the set
subjects = {'English','Physics','Chemistry','Maths'}
print("All subjects:",subjects)
# remove 'English' from a set
removedValue = subjects.discard('English')
print("Set after removing English:", subjects)
```

Output

```
All subjects: {'Physics', 'Chemistry', 'Maths', 'English'}
Set after removing English: {'Physics', 'Chemistry', 'Maths'}
```

Here, the discard() method is used to remove 'English' from the *subjects*.

Built-in Functions with Set

Following are some of the popular built-in functions used to perform different operations on a set.

Function	Description
all()	Returns True if all elements of the set are true (or if the set is empty).
any()	Returns True if any element of the set is true. If the set is empty, returns False.
enumerate()	Returns an enumerate object. It contains the index and value for all the items of the set as a pair.
len()	Returns the length (the number of items) in the set.
max()	Returns the largest item in the set.
min()	Returns the smallest item in the set.
sorted()	Returns a new sorted list from elements in the set(does not sort the set itself).
sum()	Returns the sum of all elements in the set.

Iterate Over a Set in Python

You can iterate over the elements of a set in Python using a for loop. Since sets are unordered, the order in which elements are visited during iteration is not guaranteed.

```
# Iterate over a Set
subjects = {'English','Physics','Chemistry','Maths'}
print("Iterating over the set:")
for item in subjects:
  print(item)
```

Output

```
Iterating over the set:
Maths
Physics
Chemistry
English
```

In the above examples, the for loop goes through each element in the set one by one, and you can perform operations on each element within the loop.

Finding Number of Set Elements

To determine the number of elements (also known as the "cardinality") in a Python set, the most efficient method is to use the built-in function len().

```
# Finding Number of Set Elements
subjects = {'English','Physics','Chemistry','Maths', 'Maths'}
n = len(subjects)
print("Number of subjects:",n)

Output
Number of subjects: 4
```

The len() function takes a single argument, and returns an integer representing the number of items or elements in the set. In the case of a set, len() accurately counts the number of unique elements it contains.

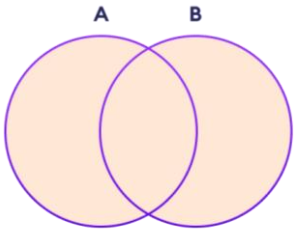
If you apply len() to an empty set, it will return 0.

Python Set Operations

Python Set provides different built-in methods to perform mathematical set operations like union, intersection, subtraction, and symmetric difference.

Union of Two Sets

The union of two sets A and B includes all the elements of sets A and B.



You can use the | operator or the union() method to perform the set union operation. For example,

```
# union of two sets
A = {1,3,5,7,9}     # first set
B = {2,4,6,8}       # second set
print("Set A :",set(A))
print("Set B :",set(B))
# perform union operation using |
```

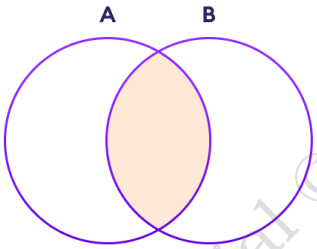
```
print('Union of set A & B using | operator:\n', A | B)
# perform union operation using union()
print('Union of set A & B using union():\n', A.union(B))
```

Output

```
Set A : {1, 3, 5, 7, 9}
Set B : {8, 2, 4, 6}
Union of set A & B using | operator:
 {1, 2, 3, 4, 5, 6, 7, 8, 9}
Union of set A & B using union():
 {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Set Intersection

The intersection of two sets A and B include the common elements between set A and B.



In Python, "&"operator or the intersection() method is used to perform the set intersection operation. For example,

```
# intersection of two sets
A = {1,3,5,7,9}             # first set
B = {1,2,3,4,5,6,8}         # second set
print("Set A :",set(A))
print("Set B :",set(B))
# intersection operation using &
print('Intersection of set A & B using & operator:\n', A & B)
# intersection operation using intersection()
print('Intersection of set A & B using intersection():\n', A.intersection(B))
```

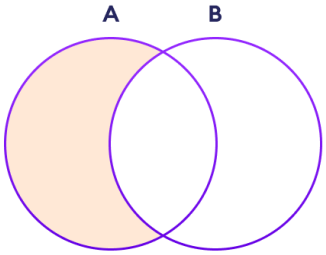
Output

```
Set A : {1, 3, 5, 7, 9}
Set B : {1, 2, 3, 4, 5, 6, 8}
Intersection of set A & B using & operator:
 {1, 3, 5}
Intersection of set A & B using intersection():
 {1, 3, 5}
```

Note: A&B and intersection() is equivalent to A ∩ B set operation.

Difference between Two Sets

The difference between two sets A and B include elements of set A that are not present on set B.



The **-** operator or the difference() is a method to perform the difference between two sets. For example,

```
# difference of two sets
A = {1,3,5,7,9}             # first set
B = {1,2,3,4,5,6,8}          # second set
print("Set A :",set(A))
print("Set B :",set(B))
# difference operation using -
print('Difference of set A & B using - operator:\n', A - B)
# difference operation using intersection()
print('Difference of set A & B using intersection():\n', A.difference(B))
```

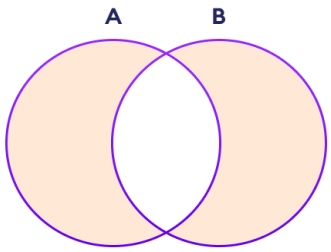
Output

```
Set A : {1, 3, 5, 7, 9}
Set B : {1, 2, 3, 4, 5, 6, 8}
Difference of set A & B using - operator:
 {9, 7}
Difference of set A & B using intersection():
 {9, 7}
```

Note: A - B and A.difference(B) is equivalent to A - B set operation.

Set Symmetric Difference

The symmetric difference between two sets A and B includes all elements of A and B without the common elements.



In Python, the **^** operator or the symmetric_difference() method is used to perform symmetric differences between two sets. For example,

```
# symmetric difference of two sets
A = {1,3,5,7,9}              # first set
B = {1,2,3,4,5,6,8}          # second set
print("Set A :",set(A))
print("Set B :",set(B))
# symmetric difference using ^ operator
print('symmetric difference of set A & B using ^ operator:\n', A ^ B)
# symmetric difference using symmetric_difference()
print('symmetric difference of set A & B using symmetric_difference():\n', A.symmetric_difference(B))
```

Output

```
Set A : {1, 3, 5, 7, 9}
Set B : {1, 2, 3, 4, 5, 6, 8}
symmetric difference of set A & B using ^ operator:
 {2, 4, 6, 7, 8, 9}
symmetric difference of set A & B using symmetric_difference():
 {2, 4, 6, 7, 8, 9}
```

Checking if two sets are equal

We can use the == operator to check whether two sets are equal or not. For example,

```
# checking two sets equal or not
A = {1,3,5,7,9}          # first set
B = {3,7,5,1,9}          # second set
C = {1,2,3,4,5}          # second set
print("Set A :",set(A))
print("Set B :",set(B))
print("Set C :",set(C))
# check two sets A & B
if A == B:
    print('Set A and Set B are equal')
else:
    print('Set A and Set B are not equal')
# check two sets A & C
if A == C:
    print('Set A and Set C are equal')
else:
    print('Set A and Set C are not equal')
```

Output

```
Set A : {1, 3, 5, 7, 9}
Set B : {1, 3, 5, 7, 9}
Set C : {1, 2, 3, 4, 5}
Set A and Set B are equal
```

```
Set A and Set C are not equal
```

In the above example, sets *A* and *B* have the same elements, so the condition A==B evaluates to True. Hence, the statement print('Set A and Set B are equal') inside the if is executed. The set *A* and C do not have the same elements, so the condition A==C evaluates to False. Hence, the statement print('Set A and Set C are not equal') inside the if is executed.

Other Python Set Methods

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with the set objects:

Method	**Description**
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
pop()	Removes and returns an arbitrary set element. Raises KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises a KeyError
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

3.8 Python Dictionary

A Python dictionary is a collection of items, similar to lists and tuples. However, unlike lists and tuples, each item in a dictionary is a **key-value** pair (consisting of a key and a value).

3.8.1 Creating a Dictionary

In Python, a dictionary is created by placing key: value pairs inside curly brackets {}, separated by commas. For example,

```
# creating a dictionary
state_capitals = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji"
}
# printing the dictionary
print("Capitals of Western Region States of India :\n", state_capitals)
```

Output

```
Capitals of Western Region States of India :
 {'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji'}
```

The state_capitals dictionary has three elements (key-value pairs), where 'Maharashtra' is the key and 'Mumbai' is the value assigned to it and so on.

Note that keys of a dictionary must be unique and immutable, such as tuples, strings, integers, etc. Immutable objects can't be changed once created. The mutable (changeable) objects such as lists cannot be used as keys. However, dictionary values can be of any data type, including mutable types like lists.

A dictionary can also be created using a Python built-in function dict().

The following example illustrates that keys of a dictionary must be immutable.

```
# keys of a dictionary must be immutable
# dictionary with integer as a key is valid
dict_int = {1:"Rohit",2:"Rahul",3:"Virat"}
print("dictionary with integer as a key is valid:\n",dict_int)
# dictionary with tuple as a key is valid
dict_tuple = {(1, 2): "Rohit Rahul", 3: "Virat"}
print("dictionary with tuple as a key is valid:\n",dict_tuple)
# dictionary with string as a key is valid
dict_str = {"Batsman": ["Rohit", "Rahul", "Virat"]}
print("dictionary with string as a key is valid:\n",dict_tuple)
# dictionary with list as a key is invalid
print("dictionary with list as a key is invalid")
dict_list = {1: "Rohit", [1, 2]: "Virat"}
```

Output

```
dictionary with integer as a key is valid:
 {1: 'Rohit', 2: 'Rahul', 3: 'Virat'}
```

```
dictionary with tuple as a key is valid:
 {(1, 2): 'Rohit Rahul', 3: 'Virat'}
dictionary with string as a key is valid:
 {(1, 2): 'Rohit Rahul', 3: 'Virat'}
dictionary with list as a key is invalid
Traceback (most recent call last):
TypeError: unhashable type: 'list'
```

In this example, we have used integers, tuples, and strings as keys for the dictionaries. When a list is used as a key, an error message occurs due to the list's mutable nature.

The keys of a dictionary must be unique. If there are duplicate keys, the later value of the key overwrites the previous value. Following example illustrate that keys of a dictionary must be unique

```
# keys of a dictionary must be unique
players = {
    "Rohit": "Batsman",
    "Rahul": "Batsman",
    "Virat": "Batsman",
    # duplicate key with a different portfolio
    "Rohit": "Captain"
}
print("Players are:",players)
```

Output

```
Players are: {'Rohit': 'Captain', 'Rahul': 'Batsman', 'Virat': 'Batsman'}
```

In this example, the key Rohit is first assigned to Batsman. However, there is a second entry where Rohit is assigned to Captain.

As duplicate keys are not allowed in a dictionary, the last entry Captain in overwrites the previous value Batsman.

Accessing Dictionary Items

The value of a dictionary item can be accessed by placing the key inside square brackets.

```
# Accessing Dictionary Items
# creating a dictionary
state_capitals = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji"
}
# access the value of keys
print("Capital of Madhya Pradesh is:", state_capitals["Madhya Pradesh"])
```

```
print("Capital of Gujrat is:",state_capitals["Gujrat"])
```

Output

```
Capital of Madhya Pradesh is: Bhopal
Capital of Gujrat is: Gandhinagar
```

Adding Items to a Dictionary

It is possible to add an item to a dictionary by assigning a value to a new key. For example,

```
# Adding Items to a Dictionary
# creating a dictionary
state_capitals = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji"
}
print("Capital of States of Western Region:\n",state_capitals)
# adding the value of keys
state_capitals["Dadra&Nagar Haveli"] = "Daman"
print("After adding capital of UT")
print("Capital of States&UTs of Western Region:\n",state_capitals)
```

Output

```
Capital of States of Western Region:
 {'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji'}
After adding capital of UT
Capital of States&UTs of Western Region:
 {'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji', 'Dadra&Nagar Haveli': 'Daman'}
```

Remove Dictionary Items

It is also possible to remove an element from a dictionary using the del statement.

Note: We can also use the pop() method to remove an item from a dictionary. To remove all items from a dictionary at once, use the clear() method.

```
# Deleting Items from a Dictionary
# creating a dictionary
state_capitals = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
```

```
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji",
  "Dadra&Nagar Haveli":"Daman"
}
print("Capital of States&UTs of Western Region:\n",state_capitals)
# delete item having "Dadra&Nagar Haveli" key
del state_capitals["Dadra&Nagar Haveli"]
print("After deleting capital of UT")
print("Capital of States of Western Region:\n",state_capitals)
```

Output

```
Capital of States&UTs of Western Region:
 {'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji', 'Dadra&Nagar Haveli': 'Daman'}
After deleting capital of UT
Capital of States of Western Region:
 {'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji'}
```

 # Deleting all items from a Dictionary

```
print("Origianl dictionary:\n",state_capitals)
# clear the dictionary
state_capitals.clear()
print("Dictionary after deleting all the items:\n",state_capitals)
```

Output

```
Original dictionary:
 {'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa':'Panaji'}
Dictionary after deleting all the items:
 {}
```

Changing Dictionary Items

Python dictionaries are mutable (changeable). It is possible to change the value of a dictionary element by referring to its key. For example,

```
# Changing Dictionary Items
# creating a dictionary
state_capitals = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji",
```

```
}
print("Origianl dictionary:\n",state_capitals)
# change the value of "Gujrat" key to "Ahmadabad"
state_capitals["Gujrat"] = "Ahmadabad"
print("Dictionary after changing item:\n",state_capitals)
```

Output

Original dictionary:

{'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji'}

Dictionary after changing item:

{'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Ahmadabad', 'Goa': 'Panaji'}

Note: You can also use the update() method to add or change dictionary items.

Iterating Through a Dictionary

A dictionary is an ordered collection of items. It maintains the order of its items. So it is possible to iterate through dictionary keys one by one using a for loop.

```
# Iterating Through a Dictionary
# creating a dictionary
state_capitals = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji",
}
print("Origianl dictionary:\n",state_capitals)

# print dictionary keys one by one
print("\nStates of Western Region:")
for state in state_capitals:
    print(state)
# print dictionary values one by one
print("\nRespective capital of States of Western Region:")
for state in state_capitals:
    capital = state_capitals[state]
    print(capital)
```

Output

Original dictionary:

{'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji'}

```
States of Western Region:
Maharashtra
Madhya Pradesh
Chhattisgarh
Gujrat
Goa

Respective capital of States of Western Region:
Mumbai
Bhopal
Raipur
Gandhinagar
Panaji
```

Finding Dictionary Length

It is possible to find the length of a dictionary by using the len() function.

```
# Finding length of dictionary
state_capitals = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji",
}
print("Origianl dictionary:\n",state_capitals)
# get dictionary's length
print("Number of items in the dictionary:", len(state_capitals))
print()
numbers = {10: "ten", 20: "twenty", 30: "thirty"}
print("Origianl dictionary:\n",numbers)
print("Number of items in the dictionary:", len(numbers))
print()
countries = {}
print("Origianl dictionary:\n",countries)
print("Number of items in the dictionary:", len(countries))
```

Output

```
Original dictionary:
 {'Maharashtra': 'Mumbai', 'Madhya Pradesh': 'Bhopal', 'Chhattisgarh': 'Raipur', 'Gujrat': 'Gandhinagar', 'Goa': 'Panaji'}
Number of items in the dictionary: 5

Origianl dictionary:
```

```
 {10: 'ten', 20: 'twenty', 30: 'thirty'}
Number of items in the dictionary: 3

Original dictionary:
 {}
Number of items in the dictionary: 0
```

Python Dictionary Methods

Here are some of the commonly used dictionary methods.

Function	Description
pop()	Removes the item with the specified key.
update()	Adds or changes dictionary items.
clear()	Remove all the items from the dictionary.
keys()	Returns all the dictionary's keys.
values()	Returns all the dictionary's values.
get()	Returns the value of the specified key.
popitem()	Returns the last inserted key and value as a tuple.
copy()	Returns a copy of the dictionary.

Testing Membership of Dictionary

It is possible to check whether a key exists in a dictionary by using the “in” and “not in” operators. The in operator checks whether a key exists; it doesn't check whether a value exists or not.

```
# Testing Membership of Dictionary
states = {
  "Maharashtra": "Mumbai",
  "Madhya Pradesh": "Bhopal",
  "Chhattisgarh": "Raipur",
  "Gujrat": "Gandhinagar",
  "Goa": "Panaji",
}
# use of in and not in operators
print("Gujrat is the member of states:","Gujrat" in states)
print("MP is the member of states:","MP" in states)
print("Goa is not the member of states:","Goa" not in states)
print("Mumbai is the member of states:","Mumbai" in states)
```

Output

```
Gujrat is the member of states: True
MP is the member of states: False
```

```
Goa is not the member of states: False
Mumbai is the member of states: False
```

Summary

Python data types define the type of value a variable can store in a program. They help the computer understand how to process and manage data.

The main data types in Python are:

- Numeric types – int, float, complex (used for numbers)
- String (str) – used to store text
- Boolean (bool) – stores True or False values
- Sequence types – list, tuple, range
- Dictionary (dict) – stores data in key–value pairs
- Set (set) – stores unique elements

Understanding data types helps in writing accurate and efficient Python programs.

Check your progress

A. Multiple Choice Questions

1. Which of the following is *not* a core data type in Python? (a) List (b) Dictionary (c) Tuple (d) Class
2. Which data type is used to store whole numbers (integers) in Python? (a) float (b) str (c) int (d) bool
3. Which of the following is an *immutable* data type in Python? (a) list (b) set (c) tuple (d) dictionary
4. What is the data type of the object x = "Hello"? (a) int (b) float (c) str (d) bool
5. Which built-in function is used to check the data type of a variable in Python? (a) datatype() (b) typeof() (c) type() (d) isinstance()
6. Which data type is used to store values in the form of key-value pairs? (a) list (b) tuple (c) set (d) dictionary
7. What will be the output of type(5 / 2) in Python? (a) <class 'int'> (b) <class 'float'> (c) <class 'str'> (d) <class 'decimal'>
8. Which of the following creates an empty *set* in Python? (a) {} (b) () (c) [] (d) set()
9. What is the output of the expression set([1, 1, 2, 3, 4, 2, 3, 4])? (a) [1, 1, 2, 3, 4, 2, 3, 4] (b) {1, 2, 3, 4} (c) {1, 1, 2, 3, 4, 2, 3, 4} (d) Invalid Syntax
10. What is the correct way to create an empty list in Python? (a) my_list = {} (b) my_list = () (c) my_list = [] (d) my_list = list() (e) Both c and d
11. Given the list my_list = [10, 20, 30, 40, 50], what is the output of print(my_list[2])? (a) 10 (b) 20 (c) 30 (d) Error
12. Which method is used to add an element to the end of a list? (a) insert() (b) add() (c) append() (d) extend()

13. Which of the following slicing operations extracts the sublist [30, 40] from my_list = [10, 20, 30, 40, 50]? (a) my_list[2:4] (b) my_list[3:5] (c) my_list[2:3] (d)my_list[1:4]
14. What is the output of len(['h', 'e', 'l', 'l', 'o'])? (a) 4 (b) 5 (c) 6 (d) Error
15. Given list1 = [1, 2, 3] and list2 = [4, 5, 6], what is the result of list1 + list2? (a) [1, 2, 3, 4, 5, 6] (b) [5, 7, 9] (c) [[1, 2, 3], [4, 5, 6]] (d) Error
16. What happens if you try to access an element beyond the bounds of a list using its index (e.g., my_list[10] if my_list has 5 elements)? (a) It returns None (b) It returns an empty list (c) It raises an IndexError (d) It returns the last element of the list
17. What is the correct way to create an empty dictionary in Python? (a) my_dict = [] (b) my_dict = () (c) my_dict = {} (d) my_dict = dict() (e) Both c and d
18. Given the dictionary my_dict = {'a': 1, 'b': 2, 'c': 3}, what is the output of my_dict['b']? (a) 'b' (b) 2 (c) 1 (d) Error
19. What happens if you try to access a dictionary key that does not exist using square brackets (e.g., my_dict['d'] if 'd' is not a key)? (a) It returns None (b) It returns an empty string (c) It raises a KeyError (d) It adds the key with a None value
20. Given my_dict = {'apple': 1, 'banana': 2, 'cherry': 3}, what is the output of my_dict.keys()? (a) dict_keys(['apple', 'banana', 'cherry']) (b) ['apple', 'banana', 'cherry'] (c) (1, 2, 3) (d) dict_keys([1, 2, 3])
21. What is the output of len({'a': 1, 'b': 2, 'c': 3})? (a) 1 (b) 2 (c) 3 (d) 6
22. What is the correct way to create an empty tuple in Python? (a) my_tuple = [] (b) my_tuple = {} (c) my_tuple = () (d) my_tuple = tuple() (e) Both c and d
23. Given the tuple my_tuple = ('apple', 'banana', 'cherry'), what is the output of my_tuple[1]? (a) 'apple' (b) 'banana' (c) 'cherry' (d) Error
24. Which operation would you use to change an element in a tuple? (a) my_tuple[1] = 'grape' (b) my_tuple.append('grape') (c) my_tuple.insert(1, 'grape') (d) You cannot change elements in a tuple. elements in a tuple
25. What happens if you try to assign a new value to an element within a tuple (e.g., my_tuple = ('a', 'b'); my_tuple = 'c')? (a) The tuple will be updated with the new value (b) It raises a TypeError (c) It returns None (d) It creates a new tuple.

B. Fill in the Blanks

1. A string (str) is a sequence of characters, enclosed in _____or _____quotes.
2. A list is an ordered, _____collection of elements, enclosed in square brackets [].
3. A tuple is an ordered, _____collection of elements, enclosed in parentheses ().
4. A dictionary (dict) stores data in _____ _pairs.
5. Dictionary keys must be _____data types, such as strings, numbers, or tuples.
6. A set is an unordered collection of _____ elements.
7. Data types that can be modified after creation are called _____ _ data types.

8. Data types that cannot be modified after creation are called _____ _data types.
9. Examples of mutable data types include _____, _____ and _____
10. Examples of immutable data types include _____, _____, and _____
11. The built-in function type() is used to check the _____ of a variable.
12. The length of a string, list, or tuple can be found using the _____ _ function.
13. _____ can be used with strings, lists, and tuples to extract a portion of the sequence.
14. Attempting to modify an element in a tuple will result in a _____
15. Negative indexing can be used to access elements from the _____ _of the list.
16. The append() method adds an element to the _____ of the list.
17. The insert() method adds an element at a _____ .
18. The remove() method removes the _____ of a specified value.
19. The del keyword can also be used to remove elements or slices from a _____
20. The clear() method removes _____ elements from a list.
21. The count() method returns the number of times a specified _____ appears in the list.
22. To create a shallow copy of a list, you can use list.copy() or _____
23. Lists can contain elements of _____ _data types.
24. Dictionary keys must be _____ _ and be _____ _ data types.
25. To access a value in a dictionary, you use its corresponding _____ _inside square brackets ([]).
26. Attempting to access a non-existent key using square brackets will raise a _____.
27. The items () method returns a view object that displays a list of _____
28. The + operator can be used to _____ _(join) two or more strings.
29. String elements (characters) can be accessed using _____
30. The in keyword can be used to check if a _____ _exists within a string.

C. State whether True or False

1. Python is a statically typed language.
2. Integers (int) in Python can store arbitrarily large whole numbers.
3. Floating-point numbers (float) can store numbers with decimal parts.
4. A string (str) is an immutable sequence of characters.
5. A list (list) is an ordered, immutable collection of items.
6. Lists can contain elements of different data types.
7. The append() method adds an element to the beginning of a list.
8. Tuples (tuples) are mutable sequences, meaning their elements can be changed after creation.

9. Dictionaries (dict) store data in key-value pairs.
10. Dictionary keys must be mutable data types.
11. Sets (set) are ordered collections of unique elements.
12. Sets automatically remove duplicate values.
13. A Boolean (bool) data type can only represent True or False.
14. None is a special data type representing the absence of a value.
15. You can use the type() function to check the data type of a variable.
16. Type conversion (type casting) is not possible in Python.
17. Empty strings, lists, tuples, and dictionaries are considered "falsy" in a Boolean context.
18. The len() function can be used to find the length of a dictionary.
19. All built-in data types in Python are immutable.
20. Custom classes created by users are generally immutable by default.

Session 4. Python Functions

In a marriage ceremony, a lot of work has to be done related to catering, decoration and other things. In general, we assign these different tasks to a particular group of people who are service providers to complete the work smoothly. For example, catering work is assigned to caterers and decoration work is assigned to decorators. This makes the process easy to manage. Similarly, in programming also we use functions to do a specific task to make our program modular and easy to read.

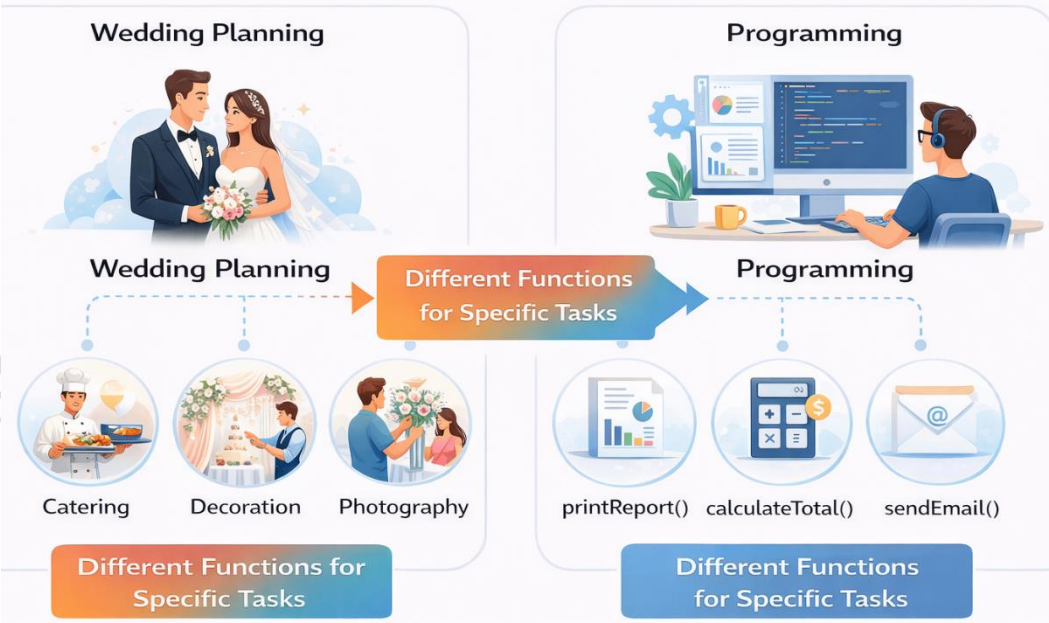


Fig. 4.1: Different tasks in a wedding ceremony

In programming, the use of function is one of the means to achieve modularity and re-usability. Function can be defined as a named group of instructions that accomplish a

specific task when it is invoked. Once defined, a function can be called repeatedly from different places of the program without writing all the codes of that function every time, or it can be called from inside another function, by simply writing the name of the function and passing the required parameters, if any. The programmer can define as many functions as desired while writing the code.

In this session, you will understand the concept of functions and the benefits of using functions. We will discuss user defined functions, flow of execution, scope of a variable and standard libraries in Python programming.

4.1 Introduction to Functions

A function is a block of code that performs a specific task. Dividing a complex problem into smaller chunks makes our program easy to understand and reuse. Suppose we need to create a program to make a circle and color it. We can create two functions to solve this problem:

1. function to create a circle
2. function to color the shape

4.1.1 Creating a Function

Example 4.1: Let's create a function greet() to greet others.

```
def greet():
    print('Hello World!')
```

The different parts of this function are as below:

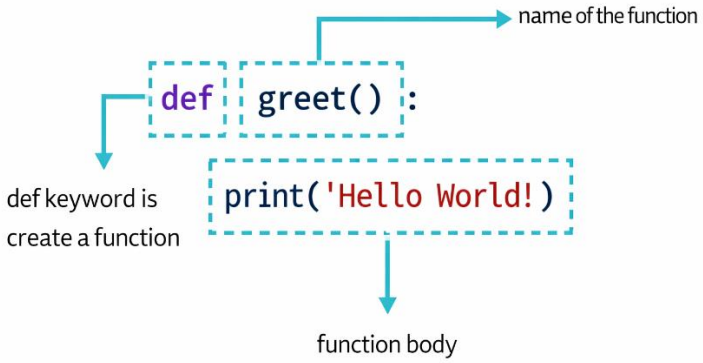


Fig. 4.2:

Here, a function greet() is created to print Hello World!

Note: When writing a function, it is important to understand indentation. Indentation are the spaces at the start of a code line.

In the above code, the print() statement is intended to show its part of the function body, distinguishing the function's definition from its body.

4.1.2 Calling a Function

In the above example, a function is declared with the name greet().

```
def greet():
    print('Hello World!')
```

The above code when executed will not produce any output. The function is created for executing code inside it. The function has to be called for its execution. The function can

be called by just mentioning the function name with opening and closing parenthesis as below.

greet()

Fig. 4.3:

Example 4.2: Let us demonstrate to call a function with the following code

```
def greet():
    print('Hello World!')
# call the function
greet()
print('Outside function')

Output
Hello World!
Outside function
```

In the above example, a function named greet() is created. The flow of control is as below:

When the function greet() is called, the program's control transfers to the function definition. The code inside the function is executed. The control of the program jumps to the next statement after the function call.

4.1.3 Python Function Arguments

Arguments are inputs given to the function. The value of the argument can be passed or mentioned itself in the function.

It is possible to pass different arguments in each call, making the function re-usable and dynamic.

In the following example, the value 'Diya' is passed to the argument of the greet() function to display the output as 'Hello Diya'

This function is again called with another argument as 'Deepak' to print Hello Deepak as shown in the below code.

```
# Calling a Function
def greet(name):
    print("Hello", name)
# pass argument
greet("Diya")
greet("Deepak")
```

```
Output
Hello Diya
Hello Deepak
```

Here, the calling of function greet() is tested by passing the different arguments in each call, making the function re-usable and dynamic.

Example 4.4: Consider another example of a function to add two numbers.

```
# function with two arguments
def add_numbers(num1, num2):
    sum = num1 + num2
    print(f"Sum of two numbers {num1} & {num2}: ", sum)
# function call with two values
add_numbers(5,4)
Output
Sum of two numbers 5 & 4:  9
```

In the above example, a function name add_numbers() is created with arguments: *num1* and *num2*.

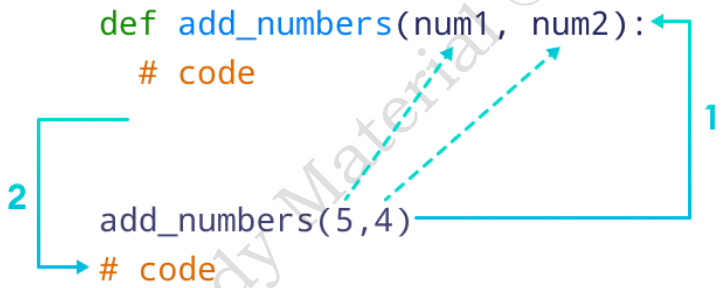


Fig. 4.4 Python Function with Arguments

4.1.3 Parameters and Arguments

Parameters: Parameters are the variables listed inside the parentheses in the function definition. The function accepts the data inside the parentheses when the function is called.

Now consider the following function display_age.

```
def display_age(age):  # age is a parameter
    print(age)
```

In this example, the display_age() function takes age as its input, whose actual value is not specified now.

Age is a parameter. The user needs to provide the specific value when the function is called.

Arguments: Arguments are the actual values that are passed to the function when it is called.

Parameters are used in the function definition and arguments are used in function call.

Arguments replace the parameters when the function executes.

print_age(21) # 21 is an argument

Here, during the function call, the argument **21** is passed to the function.

4.1.4 The return Statement

A value of the function is returned using the return statement as illustrated in

Example 4.5. Function using return statement to return the value

```
# Function using return statement
# function definition
def find_square(num):
    result = num * num
    return result

# function call
square = find_square(3)
print('Square of 3 is:', square)
square = find_square(5)
print('Square of 5 is:', square)
```

Output

Square of 3 is: 9

Square of 5 is: 25

In Example 2.4, a function named find_square() is created. The function accepts a number and returns the square of the number.

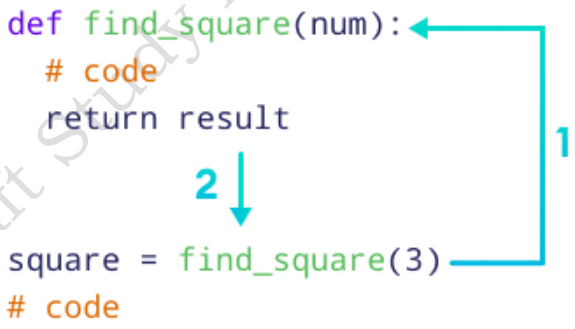


Fig. 4.5 Working of functions in Python

Note: The return statement also denotes the end of function. Any code after return is not executed.

4.1.5 The pass Statement

The pass statement is just like the continue statement. It continues the work of function without giving error even if the function body is empty. It prevents errors from empty code blocks. It's typically used where code is planned but has yet to be written.

```
def future_function():
    pass
# this will execute without any action or error
future_function()
```

4.1.6 Python Library Functions

Python provides some built-in functions that can be directly used in python programs. In such cases we don't need to create the function, but just need to call them.

Some Python library functions are:

print(): prints the string inside the quotation marks

sqrt(): returns the square root of a number

pow(): returns the power of a number

These library functions are defined inside the module, and to use them, you must include the math module inside our program. For example, sqrt() is defined inside the math module.

Example 4.6 illustrates the use of Python library function.

```
# Using Python Library Function
import math
# sqrt computes the square root
square_root = math.sqrt(4)
print("Square Root of 4 is",square_root)
# pow() comptes the power
power = pow(2, 3)
print("2 to the power 3 is",power)
```

Output

```
Square Root of 4 is 2.0
2 to the power 3 is 8
```

4.1.7 User Defined Function Vs Standard Library Functions

In Python, functions are divided into two categories: *user-defined functions* and *standard library functions*.

Standard Library Functions – There are a large number of functions already available in Python under the standard library. We can directly call these functions in our program without defining them. Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of a library, it can be termed as library functions.

User-Defined Functions – Functions that we define ourselves to do certain specific task are referred to as user-defined functions. They are designed for specific tasks to be performed as per our requirement. They are not part of Python's standard toolbox. A user can create the function as per their need.

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes the program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.

3. Programmers working on large projects can divide the workload by making different functions.

Example 4.7: Following program illustrates how to write a user defined function to add two numbers and display their sum as a result.

```
# User defined function to add 2 nos. and display sum
def add():
    num1=int(input("Enter first no. : "))
    num2=int(input("Enter second no. : "))
    sum=num1+num2
    print("The sum of",num1,"and",num2,"is :",sum)

# function call
add()
```

Output

```
Enter first no. : 5
Enter second no. : 6
The sum of 5 and 6 is : 11

Enter first no. : 27
Enter second no. : 56
The sum of 27 and 56 is : 83
```

In the above program, add() is a user-defined function that takes two integer numbers as input, calculates the sum of two numbers and displays it. To execute this function, it is required to call this function by using function name (). The function add() is being called in the last line of code

The *def* is the keyword used to define a function and the function name is written following the def keyword. After execution, it creates a new function object and assigns it a new name. It can be used inside any control structures like if else.

Example 4.8 illustrates this.

```
# Using function inside the if...else
def test():
    a=int(input("Enter a value : "))
    if a>0:
        print("Positive")
    else:
        print("Negative")
# Function call
test()
```

Output

```
Enter a value : 5
The number is Positive
```

```
Enter a value : -1
The number is Negative
```

Assignment 4.1

1. Write a program to find the maximum of three numbers using a user-defined function.
2. Write a program to Find Factorial of Number using a user-defined function.
3. Write a program to print the sum of digits of a user entered number using a user-defined function.
4. Write a program to print the day name by reading the day number from the user using a user-defined function.

Example 4.9 : Let's look at an example.

```
# calling function with arguments
def greet(name, message="Hello"):
    print(message, name)

# calling function with both arguments
greet("Students", "Good Morning")

# calling function with only one argument
greet("Diya")
```

Output

```
calling function with both arguments
Good Morning Students
calling function with only one argument
Hello Diya
```

4.2 Python Function Arguments

An argument is a value that is accepted by a function. In the above example, the values are accepted from the user within the function itself, but it is also possible for a user defined function to receive values when it is called. An argument is a value passed to the function during the function call which is received in the corresponding parameter defined in the function header.

Example 4.10: Write a python code to add two numbers using a function with arguments.

```
# Sum of 2 numbers by using function
# with specified arguments
def add_numbers(a, b):
    sum = a + b
    print('Sum using function with arguments',a,b,'=',sum)
```

```
# Function call with arguments 4,5
add_numbers(4,5)

# Function call with arguments 4,-5
add_numbers(4,-5)
```

Output

```
Sum using function with arguments 4 5 = 9
Sum using function with arguments 4 -5 = -1
```

In the above example, the function add_numbers() takes two parameters: a and b. The function, add_numbers(4, 5) specifies that parameters a and b will get values **4** and **5** respectively. The function, add_numbers(4, 5) specifies that parameters a and b will get values **-4** and **5** respectively.

4.2.1 Function Argument with Default Values

Python allows assigning a default value to the parameter. Default arguments are used when no explicit values are passed to these parameters during a function call. The = operator is used to provide default values.

Example 4.11: For example, the above code can be modified as below.

```
# Sum of 2 nos. by using function
# argument with default values
def add_numbers(a = 7,  b = 8):
    sum = a + b
    print('Sum = ',sum)

print('default arguments a=7, b=8')
# function call with two arguments
print('When both arguments a=2 & b=3 provided to function')
add_numbers(2, 3)

# function call with one argument
print('When first arg a=2 provided, value of b=8 taken by default')
add_numbers(a = 2)

# function call with no arguments
print('When no arg provided, value of a=7 & b=8 taken by default:')
add_numbers()
```

Output

```
default arguments a=7, b=8
When both arguments a=2 & b=3 provided to function
Sum =  5
When first arg a=2 provided, value of b=8 taken by default
```

```
Sum =  10
When no arg provided, value of a=7 & b=8 taken by default:
Sum =  15
```

In this example the default values **7** and **8** are provided for parameters a and b respectively. Let us see how this program works for three conditions

1. add_number(2, 3) : Both values are passed during the function call. Hence, these values are used instead of the default values.

2. add_number(2) : Only one value is passed during the function call. So, according to the positional argument **2** is assigned to argument a, and the default value is used for parameter b.

3. add_number() : No value is passed during the function call. Hence, default value is used for both parameters a and b.

Example 4.12: Consider another example to convert the denominator of the division into proper fraction using a user defined function.

```
# Program to convert denominator of division into
# proper fraction using user defined function
def mixedFraction(num, deno = 1):
    remainder = num % deno
    if remainder != 0:
        quotient = int(num/deno)
        print("The mixed fraction=    ",quotient,'(',remainder,'/',deno,')')
    else:
        print("It evaluates to whole number")
# Function ends
num = int(input("Enter the numerator : "))
deno = int(input("Enter the denominator : "))
print("The number is :",num, '/',deno)
if num > deno:              # Check for proper fraction
    mixedFraction(num,deno) # Function call
else:
    print("It is a proper fraction")
```

Output

```
Enter the numerator : 20
Enter the denominator : 3
The number is : 20 / 3
The mixed fraction =  6 ( 2 / 3 )

Enter the numerator : 20
Enter the denominator : 2
The number is : 20 / 2
It evaluates to whole number
```

```
Enter the numerator : 6
Enter the denominator : 20
The number is : 6 / 20
It is a proper fraction
```

In the above program, the denominator entered is 3, which is passed to the parameter *"deno"* so the default value of the argument *deno* is overwritten. Let us consider the following function call: mixedFraction(9). Here, num will be assigned 25 and *deno* will use the default value 1.

A function argument can also be an expression, such as

mixedFraction(num+5, deno+5)

In such a case, the argument is evaluated before calling the function so that a valid value can be assigned to the parameter. The parameters should be in the same order as that of the arguments.

The default parameters must be the trailing parameters in the function header that means if any parameter is having default value then all the other parameters to its right must also have default values. For example,

```
def mixedFraction(num,deno = 1)
def mixedFraction(num = 2,deno = 1)
```

Let us consider few more function definition headers:

```
def calcInterest(principal = 1000, rate, time = 5):
```

Above header is incorrect as default must be the last #parameter. So, the correct function header should be as follows.

```
def calcInterest(rate, principal = 1000, time = 5):
```

One more point is important to note that a function header cannot have expressions. Therefore, following function headers will give error:

```
def mixedFraction(num+5,deno):
def mixedFraction(num+5,deno+5):
```

4.2.2 Python Keyword Argument

In keyword arguments, arguments are assigned based on the name of the arguments. For example,

```
# Python Keyword Argument
def display_info(first_name, last_name):
   print('First Name:', first_name)
   print('Last Name:', last_name)

display_info(last_name = 'Deepak', first_name = 'Diya')
```

Output

```
First Name: Diya
Last Name: Deepak
```

Here, notice the function call, names are assigned to arguments during the function call.

Hence, first_name in the function call is assigned to first_name in the function definition. Similarly, last_name in the function call is assigned to last_name in the function definition. In such cases, the position of arguments doesn't matter.

4.2.3 Python Function with Arbitrary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function. To handle this kind of situation, arbitrary arguments are used in Python.

Arbitrary arguments allow passing a varying number of values during a function call. An asterisk (*) is used before the parameter name to denote this kind of argument. For example,

```
# Python Function With Arbitrary Arguments
# program to find sum of multiple numbers

def find_sum(*numbers):
    result = 0

    for num in numbers:
        result = result + num

    print("Sum = ", result)

# function call with 3 arguments
print('function call with 3 arguments 1,2,3')
find_sum(1, 2, 3)

Output
# function call with 2 arguments
print('function call with 2 arguments 4,9')
find_sum(4, 9)
function call with 3 arguments 1,2,3
Sum =  6
function call with 2 arguments 4,9
Sum =  13
```

In the above example, the function find_sum() is created to accept arbitrary arguments. Here, one can call the same function with different arguments.

Note: After getting multiple values, numbers behave as an array so we are able to use the for loop to access each value.

4.2.4 String as Parameters

In the above program only numeric types of arguments are passed while calling a function. However, it may also require passing string values as an argument, as illustrated in *Example 4.13.*

Example 4.13: Python Function with string parameters

```
# Python Function with string parameters
def display_info(first_name, last_name):
    print('First Name:', first_name)
    print('Last Name:', last_name)
    print('Full Name:', first_name, last_name)

display_info('Rajnish', 'Kumar')
```

Output

```
First Name: Rajnish
Last Name: Kumar
Full Name: Rajnish Kumar
```

Here, the names to arguments are assigned during the function call. Hence, first_name in the function call is assigned to first_name in the function definition. Similarly, last_name in the function call is assigned to last_name in the function definition. In such cases, the position of arguments doesn't matter.

Example 4.14: Consider another example to display the full name by taking the input first name and last name using concatenation of two strings.

```
# Program to concatenate two strings parameters
# Function Start
def full_name(first_name, last_name):
    full_name = first_name + last_name
    print('Full Name : ',full_name)
# Function End
first_name = input('Enter First Name : ')
last_name = input('Enter Last Name : ')

# Function Call
full_name(first_name, last_name)

Output
Enter First Name : Diya
Enter Last Name : Deepak
Full Name :  DiyaDeepak
```


4.3 Scope of a variable

A variable defined inside a function cannot be accessed outside it. Every variable has a well-defined accessibility. The part of the program where a variable is accessible can be defined as the scope of that variable.

In Python, you can declare variables in three different scopes: *local scope, global,* and *nonlocal scope*.

A variable scope specifies the region where we can access a variable. For example,

```
def add_numbers():
```

```
sum = 5 + 4
```

Here, the *sum* variable is created inside the function, so it can only be accessed within it (local scope). This type of variable is called a *local* variable.

Based on the scope, Python variables are classified into three types:

1. Local Variables
2. Global Variables
3. Nonlocal Variables

4.3.1 Local Variables

When you declare variables inside a function, these variables will have a local scope (within the function). It can be accessed only in the function or a block where it is defined. It exists only till the function executes.

These types of variables are called local variables. For example,

Example 4.16: # Illustrate the scope of local variable

```
# Illustrate the scope of local variable
def greet():
    # local variable
    message = 'Hello'
    print('Local:', message)
greet()
print('Local',message)

Output
Local: Hello
Local
NameError: name 'message' is not defined
```

Here, the *message* variable is local to the greet() function, so it can only be accessed within the function. When you try to access it outside the greet() function, it gives an error.

4.3.2 Global Variables

In Python, a variable declared outside of the function or in global scope is known as a global variable. A global variable can be accessed inside or outside of the function. Any change made to the global variable will impact all the functions in the program where that variable can be accessed.

Example 4.17: Illustrate the scope of global variable

```
# Illustrate the scope of global variable
# declare global variable
message = 'Hello'
def greet():
    # declare local variable
    print('Local', message)
greet()
```

```
print('Global', message)
```

Output

Local Hello
Global Hello

This example illustrates that when you declare the *message* variable globally, it is possible to access it within the function as well outside the function.

4.3.3 Nonlocal Variables

In Python, the nonlocal keyword is used within nested functions to indicate that a variable is not local to the inner function, but rather belongs to an enclosing function's scope.

This allows you to modify a variable from the outer function within the nested function, while still keeping it distinct from global variables.

Example 4.18: Illustrate the scope of nonlocal variable

```
# outside function
def outer():
    message = 'local'
    # nested function
    def inner():
        # declare nonlocal variable
        nonlocal message
        message = 'nonlocal'
        print("inner:", message)
    inner()
    print("outer:", message)
outer()
```

Output

inner: nonlocal
outer: nonlocal

In the above example, there is a nested inner() function. The inner() function is defined in the scope of another function outer().

The nonlocal keyword is used to modify the message variable from the outer function within the nested function.

Note : If you change the value of a nonlocal variable, the changes appear in the local variable.

4.3.4 Global Keyword

In Python, the global keyword allows modifying the variable outside of the current scope. It is used to create a global variable and make changes to the variable in a local context.

Access and Modify Python Global Variable

First let's try to access a global variable from the inside of a function,

Example 4.19: Accessing a global variable from the inside of a function

```
# Accessing a global variable inside a function
c = 1 # global variable
def add():
    print('value of global variable:',c)
    print('incrementing c by 2:')
    print(c=c+2)

add()

Output:
value of global variable: 1
incrementing c by 2:

TypeError: 'c' is an invalid keyword argument for print()
```

Here, a global variable is accessed inside a function. But if you try to modify the global variable inside a function, it gives an error.

This is because we can only access the global variable but cannot modify it from inside the function.

The solution for this is to use the global keyword.

Example 4.20 illustrates how to change a global variable inside a function using the global keyword.

Example 4.20: Changing global variable inside a function using global keyword.

```
# Changing global variable inside
# a function using global keywords.
c = 1          # global variable
def add():
    global c    # use of global keyword
    print('value of global variable c =',c)
    print('incrementing c by 2:')
    c = c + 2   # increment c by 2
    print('incremented value of c =',c)
add()

Output
value of global variable c = 1
incrementing c by 2:
incremented value of c = 3
```

In the above example, *c* is defined as a global variable and a global keyword is used for c inside the add() function. Then, the value of c is incremented by 2. Since the global

keyword is used for c inside the function, c = c + 2 works and the value of c is incremented 2 and the new value 3 is assigned to variable c.

Rules of global Keyword

The basic rules for global keyword in Python are:

a) Creating a variable inside a function is treated as local by default.

b) When a variable is defined outside the function, it is global by default. You don't have to use the global keyword.

c) The global keyword is used to modify a global variable inside a function.

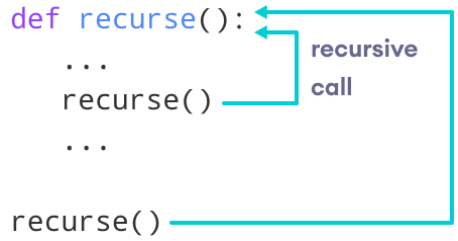
d) Using the global keyword outside a function has no effect.

4.4 Recursion

Recursion is the process of defining something in terms of itself. A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

4.4.1 Recursive Function

In Python, a function can call other functions. It is even possible for the function to call itself. When a function calls itself, it is called as recursive functions. The following illustration shows the working of a recursive function.



The factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1*2*3*4*5*6 = 720. Let us write a program to find the factorial of a number using recursion.

Example 4.21: Factorial of a number using recursion

```
# Factorial of a number using recursion
def factorial(x):
# This is a recursive function to
# find the factorial of an integer
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
num = 3
print("The factorial of", num, "is", factorial(num))
```

Output

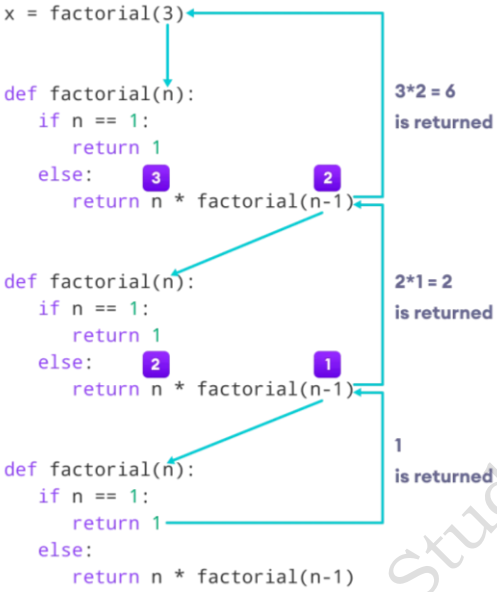
The factorial of 3 is 6

In the above example, factorial() is a recursive function as it calls itself. When you call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one. This recursive call is explained in the following steps.

```
factorial(3)          # 1st call with 3
3 * factorial(2)      # 2nd call with 2
3 * 2 * factorial(1)  # 3rd call with 1
3 * 2 * 1             # return from 3rd call as number=1
3 * 2                 # return from 2nd call
6                     # return from 1st call
```

Figure 4.6 shows a step-by-step process of the working of code.



The recursion ends when the number reduces to 1. This is called the base condition.

Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

The Python interpreter limits the depths of recursion to help avoid infinite recursions, resulting in stack overflows.

By default, the maximum depth of recursion is 1000. If the limit is crossed, it results in RecursionError, as shown below.

```
def recursor():
    recursor()
recursor()
```

Output

```
RecursionError: maximum recursion depth exceeded
```

Advantages of Recursion

a) Recursive functions make the code look clean and elegant.

b) A complex task can be broken down into simpler sub-problems using recursion.

c) Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

a) Sometimes the logic behind recursion is hard to follow through.

b) Recursive calls are expensive (inefficient) as they take up a lot of memory and time.

c) Recursive functions are hard to debug.

Example 4.22: Write a program to calculate the factorial of a number using recursion.

```
# Program to find the factorial
# by taking input from the user
num = int(input("Enter a number: "))
factorial = 1
# check if the number is negative, positive or zero
if num < 0:
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1,num + 1):
       factorial = factorial*i
   print("The factorial of",num,"is",factorial)
```

Output

```
Enter a number: 1
The factorial of 1 is 1

Enter a number: 2
The factorial of 2 is 2

Enter a number: 3
The factorial of 3 is 6

Enter a number: 5
The factorial of 5 is 120
```

You can run this program for different input as shown below.

Example 4.23: Write a program to Find Sum of Natural Numbers Using Recursion

```
# program to find the sum of natural
# using recursive function
```

```
n = int(input("Enter a number: "))
def recur_sum(n):
   if n <= 1:
      return n
   else:
      return n + recur_sum(n-1)

if n < 0:
    print("Enter a positive number")
else:
    print(f"The sum of first {n} natural nos. is",recur_sum(n))
```

Output

```
Enter a number: 1
The sum of first 1 natural nos. is 1

Enter a number: 2
The sum of first 2 natural nos. is 3

Enter a number: 10
The sum of first 10 natural nos. is 55

Enter a number: 100
The sum of first 100 natural nos. is 5050

Enter a number: -10
Enter a positive number
```

Example 4.24: Write a program to display Fibonacci Sequence Using Recursion.

A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8....

The first two terms are 0 and 1. All other terms are obtained by adding the preceding two terms.This means to say the nth term is the sum of $(n-1)^{th}$ and $(n-2)^{th}$ term.

```
# program to display the Fibonacci sequence
def recur_fibo(n):
   if n <= 1:
      return n
   else:
      return(recur_fibo(n-1) + recur_fibo(n-2))
nterms = int(input("Enter the number of terms : "))
# check if the number of terms is valid
if nterms <= 0:
   print("Plese enter a positive integer")
```

```
else:
   print(f'Fibonacci sequence upto {nterms} terms is:")
   for i in range(nterms):
      print(recur_fibo(i))

Output
Enter the number of terms : 1
Fibonacci sequence upto 1 terms is:
0

Enter the number of terms : 2
Fibonacci sequence upto 2 terms is:
0
1

Enter the number of terms : 3
Fibonacci sequence upto 3 terms is:
0
1
1

Enter the number of terms : 5
Fibonacci sequence upto 5 terms is:
0
1
1
2
3

Enter the number of terms : -10
Plese enter a positive integer
```

In this program, the number of terms to be displayed is stored in *nterms*.

A recursive function recur_fibo() is used to calculate the nth term of the sequence. A for loop is used to iterate and calculate each term recursively.

4.5 Python Modules

A big program may contain a number of lines of code. Instead of putting everything in a single file, you can use modules to separate codes in separate files as per their functionality. This makes the code organized and easier to maintain.

Module is a file that contains code to perform a specific task. A module may contain variables, functions, classes etc.

Let us create a module. Type the following and save it as example.py

```
# Python Module addition
def add(a, b):
   result = a + b
   return result
```

Here, function add() is defined inside a module named example. The function takes in two numbers and returns their sum.

4.5.1 Importing modules in Python

It is possible to import the definitions inside a module to another module or the interactive interpreter in Python using import keyword.

To import our previously defined module example, we type the following in the Python prompt.

```
import example
```

This does not import the names of the functions defined in example directly in the current symbol table. It only imports the module name example there.

Using the module name it is possible to access the function using the dot . operator. For example:

```
example.add(4,5) # returns 9
```

4.5.2 Import Python Standard Library Modules

Python standard library contains over 200 modules. You can import them according to your needs.

Suppose you want to get the value of pi, for that first you need to import the math module and then use math.pi. For example,

```
# import standard math module
import math

# use math.pi to get value of pi
print("The value of pi is", math.pi)
```

Output

```
The value of pi is 3.141592653589793
```

4.5.3 import with Renaming

In Python, it is possible to import a module by renaming it. For example,

```
# import module by renaming it
import math as m

print(m.pi)

# Output: 3.141592653589793
```

Here, the math module is renamed as m. This can save typing time if the code is so long. When you rename the module the math.pi will not work, instead you have to use it as m.pi as shown in the above example.

4.5.4 Python from...import statement

It is also possible to import specific names from a module without importing the module as a whole. For example, to import only the pi attribute from the math module, write a code as below.

```
# import only pi from math module
from math import pi

print(pi)

# Output: 3.141592653589793
```

4.5.5 Import all names

In Python, you can also import all names (definitions) from a module using the following construct:

```
# import all names from the standard module math
from math import *

print("The value of pi is", pi)
```

Here, all the definitions from the math module are imported. This includes all names visible in the scope except those beginning with an underscore (private definitions).

Importing everything with the asterisk (*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

Summary

Python functions are reusable blocks of code that perform a specific task. They help in organizing programs into smaller, manageable parts and reduce repetition of code.

A function is defined using the def keyword. It can take parameters (inputs) and may return a value using the return statement.

Functions improve code readability, reusability, and efficiency. Python provides both built-in functions (like print(), len()) and user-defined functions created by programmers.

Check Your Progress

A. Multiple Choice Questions

1. Which keyword is used to define a function in Python? (a) function (b) def (c) func (d) define
2. What is the purpose of the return statement in a Python function? (a) To print a statement (b) To exit the function and pass back a value (c) To define a variable (d) To assign a value to a variable

3. What happens if a return statement is not used inside a function? (a) The function will return 0. (b) The function will return None (c) The function will return an arbitrary value (d) The function will raise an error
4. Which of the following describes a function that calls itself? (a) A built-in function (b) A user-defined function (c) A recursive function (d) A lambda function
5. Which type of argument allows a function to accept any number of positional arguments? (a) Default arguments (b) Keyword arguments (c) *args (d) **kwargs
6. What is the scope of a variable defined inside a function? (a) Global scope (b) Local scope (c) Enclosing scope (d) Built-in scope
7. What is a function defined inside a class called? (a) A module (b) A function (c) A method (d) A nested function
8. How do you call a function named my_function without any arguments? (a) call my_function() (b) my_function (c) my_function() (d) run my_function()
9. What is the purpose of the global keyword in Python? (a) To create a local variable inside a function (b) To access a global variable from inside a function (c) To modify a global variable from inside a function (d) Both b and c
10. What is the primary benefit of using functions in Python? (a) Reducing code duplication (b) Improving code clarity and readability (c) Breaking down complex problems into smaller parts (d) All of the mentioned
11. Which of the following is *not* a type of function argument in Python? (a) Positional arguments (b) Keyword arguments (c) Default arguments (d) Reference arguments
12. Which of the following refers to mathematical function? (a) sqrt (b) rhombus (c) add (d) minus

B. Fill in the Blanks

1. A function is a block of_____ code that performs a specific task.
2. The keyword used to define a function in Python is _____ .
3. Functions help organize code into manageable _____ and promote code _____.
4. Data passed into a function are known as _____ .
5. If a function does not explicitly return a value, it implicitly returns _____ .
6. A variable defined inside a function has _____ scope.
7. The _____ statement is used to exit a function and pass back a value to the caller.
8. The _____ keyword is used inside a function to modify a variable defined outside the function's scope.
9. When you call a function, you pass _____ to it.
10. A function defined inside a class is called a _____ .

C. State whether True or False

1. Function header always ends with a semicolon (;).
2. Functions in Python are defined using the func keyword.

3. A Python function can have multiple return statements, but only one will be executed.
4. If a function does not contain an explicit return statement, it will automatically return the value 0.
5. A local variable within a function can be accessed directly from outside that function.
6. The global keyword is used inside a function to create a new global variable.
7. *args collects a variable number of positional arguments into a list inside the function.
8. Default arguments must be placed after all non-default arguments in a function definition.
9. Every recursive function must have a base case to avoid infinite recursion.
10. Recursive functions always run faster than their iterative counterparts.
11. The nonlocal keyword is used to modify a global variable from within a nested function.
12. Functions help in improving code organization and modularity.
13. When calling a function, you must always provide arguments in the same order as the parameters are defined in the function signature.
14. A function can be passed as an argument to another function in Python.
15. Using many global variables is generally considered a good practice in Python programming.

D. Programming Questions

- Identify the errors if any in the following code.

(a) def create (text, freq):

```
    for i in range (1, freq):
    print text
    create (5)        #function call
```

(b) from math import sqrt,ceil

```
    def calc ():
    print cos (0)
    calc ()   #function call
```

(c) mynum = 9 def add9():

```
    mynum = mynum + 9
    print mynum
    add9()  #function call
```

(d) def findValue(val) = 1.1, val2, val3):

```
    final = (val2 + val3)/ vall
    print(final)
```

```
findvalue ()     #function call
```

(e) def greet ():

```
return ("Good morning")
greet () = message     #function call
```

1. Write a program to check the divisibility of a number by 7 that is passed as a parameter to the user defined function.
2. Write a program that uses a user defined function that accepts name and gender (as M for Male, F for Female) and prefixes Mr./Ms. on the basis of the gender.
3. Write a Program that uses two user defined functions to convert temperatures to and from Celsius, Fahrenheit and Fahrenheit to Celsius.
4. Write a program that has a user defined function to accept the coefficients of a quadratic equation in variables and calculates its determinant. For example: if the coefficients are stored in the variables a, b, c then calculate the determinant as b^2-4ac. Write the appropriate condition to check determinants on positive, zero and negative and output appropriate results.
5. ABC School has allotted unique token IDs from (1 to 600) to all the parents for facilitating a lucky draw on the day of their annual day function. The winner would receive a special prize. Write a program using Python that helps to automate the task. (Hint: use random module)
6. Write a program that implements a user defined function that accepts Principal Amount, Rate, Time, Number of Times the interest is compounded to calculate and displays compound interest. (Hint: CI = $P\left(1+\frac{R}{N}\right)^{NT}$
7. Write a program that has a user defined function to accept 2 numbers as parameters, if number 1 is less than number 2 then numbers are swapped and returned, i.e., number 2 is returned in place of number 1 and number 1 is reformed in place of number 2, otherwise the same order is returned.
8. Write a program that contains user defined functions to calculate area, perimeter or surface area whichever is applicable for various shapes like square, rectangle, triangle, circle and cylinder. The user defined functions should accept the values for calculation as parameters and the calculated value should be returned. Import the module and use the appropriate functions.
9. Write a program that creates a GK quiz consisting of any five questions of your choice. The questions should be displayed randomly. Create a user defined function score () to calculate the score of the quiz and another user defined function remark (score value) that accepts the final score to display remarks as follows:

Marks	**Remarks**
5	Outstanding
4	Excellent
3	Good
2	Read more to score more

1	Needs to take interest
0	Take it seriously.

APPENDIX

Table 1: Commonly used functions in math module

Function Syntax	Arguments	Returns	Example Output

math.ceil(x)	x may be an integer or floating-point number	ceiling value of x	>>> math.ceil(-9.7)
			-9
			>>> math.ceil (9.7)
			10
			>>> math.ceil(9) 9

math.floor(x)	x may be an integer or floating-point number	floor value of x	>>> math.floor(-4.5)
			-5
			>>> math.floor(4.5)
			4
			>>> math.floor(4)
			4

math.fabs(x)	x may be an integer or floating-point number	absolute value of x	>>> math.fabs(6.7)
			6.7
			>>> math.fabs(-6.7)
			6.7
			>>> math.fabs(-4) 4.0

math.factorial(x)	x is a positive integer	factorial of x	>>> math.factorial(5) 120

math.fmod(x,y)	x and y may be an integer or floating-point number	x % y with sign of x	>>> math.fmod(4,4.9)
			4.0
			>>> math.fmod(4.9,4.9)
			0.0
			>>> math.fmod(-4.9,2.5)
			-2.4
			>>> math.fmod(4.9,-4.9)
			0.0

math.gcd(x,y)	x, y are positive integers	gcd (greatest common divisor) of x and y	>>> math.gcd(10,2) 2

math.pow(x,y)	x, y may be an integer or floating-point number	x^y (x raised to the power y)	>>> math.pow(3,2)
			9.0
			>>> math.pow(4,2.5)
			32.0
			>>> math.pow(6.5,2)
			42.25
			>>> math.pow(5.5,3.2)
			233.97

math.sqrt(x)	x may be a positive integer or floating-point number	square root of x	>>> math.sqrt(144)
			12.0
			>>> math.sqrt(.64)
			0.8

math.sin(x)	x may be an integer or floating-point number in radians	sine of x in radians	>>> math.sin(0)
			0
			>>> math.sin(6)
			-0.279

Table 2. Commonly used functions in random module

Function Syntax	**Argument**	**Return**	**Example Output**
random. random()	No argument (void)	Random Real Number (float) in the range 0.0 to 1.0	>>> random. random() 0.65333522
random. randrange(x,y)	x and y are positive integers signifying the start and stop value	Random integer between x and y	>>> random.randrange(2,7) 2
random. randrage(y)	y is a positive integer signifying the stop value	Random integer between 0 and y	>>> random.randrange(5) 4

Table 3. Some of the function available through statistics module

Function Syntax	**Argument**	**Return**	**Example Output**
statistics.mean(x)	x is a numeric sequence	arithmetic mean	>>> statistics. mean([11,24,32,45,51]) 32.6
statistics.median(x)	x is a numeric sequence	median (middle value) of x	>>>statistics. median([11,24,32,45,51]) 32
statistics.mode(x)	x is a sequence	mode (the most repeated value)	>>> statistics. mode([11,24,11,45,11]) 11 >>> statistics. mode(("red","blue","red")) 'red'

Module 3. Data Literacy

Module Overview

In today's digital age, data has become as important as reading and writing. Every action we take — from browsing social media, ordering food online, to checking our health apps — creates data. But data in its raw form (numbers, words, or images) is not useful unless we can understand, analyze, and apply it. This ability is called Data Literacy. Being data literate means making informed, fair, and responsible decisions in both personal and professional life.

Data Literacy empowers us to:

- Distinguish between data and information.
- Identify different types and sources of data.
- Ensure data quality through cleaning and accuracy.
- Use statistics and visualization to interpret patterns.
- Recognize ethical issues and biases in data and AI systems.

Learning Outcomes

After completing this module, you will be able to:

- Explain the role of data in society and understand how data influences decision-making in business, government, healthcare, and education.
- Identify different types and sources of data such as structured and unstructured data, primary and secondary data.
- Understand the importance of data quality and perform basic data cleaning tasks like removing errors, duplicates, and missing values.

- Apply basic statistical concepts such as mean, median, mode, range, and graphs for simple data analysis.
- Recognize the importance of accurate and reliable data for effective analysis and decision-making.

Module Structure

Session 1. Role of Data in Society

Session 2. Types and Sources of Data

Session 3. Data Quality and Data Cleaning

Session 4. Statistics for Data Analysis

Session 1. Role of Data in Society

Riya a school student, sees a viral social media post claiming that drinking a specific herbal tea cure COVID-19. The post has thousands of likes and shares.

Instead of blindly believing it, Riya applies her data literacy skills:

1. She questions the data – Where did this claim come from? Is there any source mentioned?
2. She investigates the source – She finds that the website is not a verified health organization.
3. She looks for patterns – She checks if any reputable sources like WHO or CDC are reporting the same thing (they're not).
4. She verifies with reliable data – She finds a WHO report clearly stating that no such herbal tea has been proven effective against COVID-19.

She makes a data-informed decision – She avoids sharing the post and reports it as misinformation. Figure 1.1 illustrates the data literacy process.

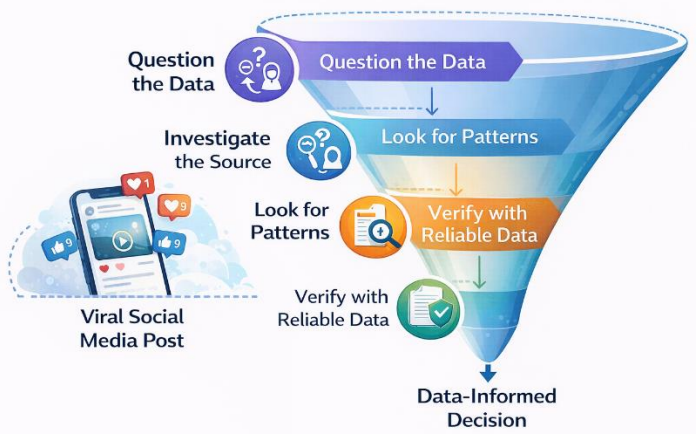


Fig. 1.1: Data Literacy Process

Riya's ability to understand, evaluate, and interpret data helped her avoid misinformation and act responsibly online. This is data literacy in action — not just

about numbers, but about making smart, safe decisions in everyday life using data. In this session we are going to discuss the role of data in society.

1.1 Introduction to Data and Information

Data refers to raw, unprocessed facts and figures. It does not carry any meaning on its own until it's analyzed or organized.

Example: A list like 34, "blue", 98, "Mumbai" is data. Without context, we don't know what it means. Figure 1.2 shows data context and meaning.

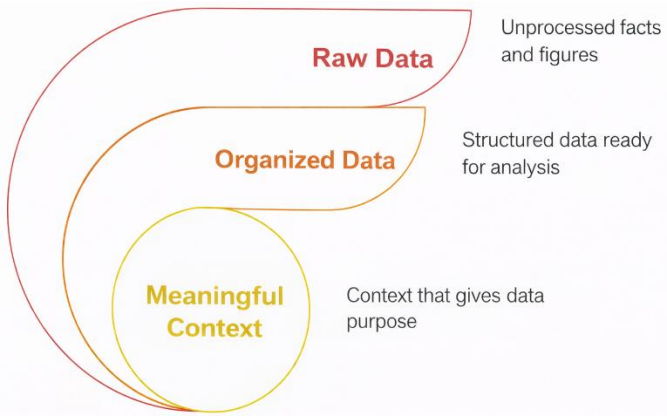


Fig.1.2: Data Context

Types of Data

There are different types of data. In our next session we will talk about it in detail. Let us understand some basic information here.

Qualitative Data: It is descriptive data. For example: Colors, names, emotions such as, "happy", "blue" is a qualitative data.

Quantitative Data: It is a numerical data. For example, numbers, measurements such as, 10 kg, 98 marks is a quantitative data. Figure 3 shows types of data.

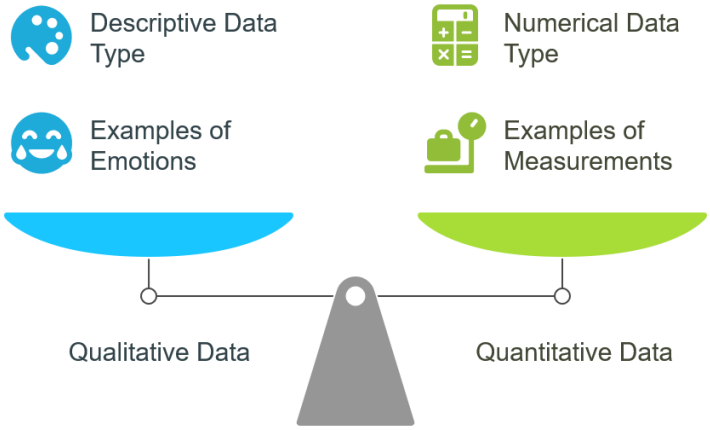


Fig. 1.3: Types of Data

What is Information?

Information is processed, organized, or structured data that has meaning and is useful for decision-making. For example, if we say:

"Ravi scored 98 marks in Mathematics and lives in Mumbai," this is information because the data has been given context and meaning. Figure 1.4 shows illustration.

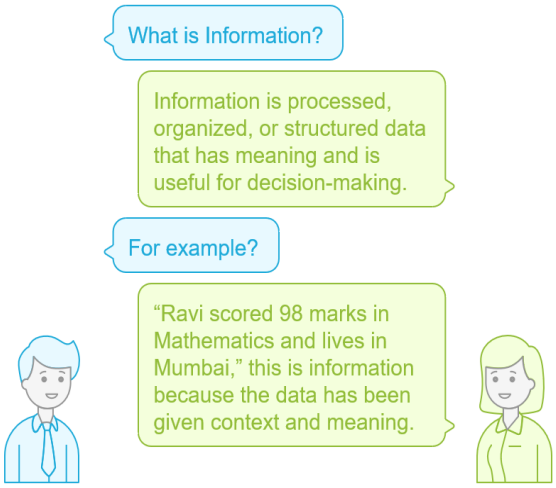


Fig. 1.4: Defining Information

Key Differences Between Data and Information

Aspect	Data	Information
Meaning	Raw facts	Processed data with meaning
Context	No context	Has context and relevance
Usage	Not directly useful	Useful for decision-making
Example	"Red", "25", "Banana"	"The 25-year-old ate a red banana"

In today's digital world, understanding the difference between data and information is the first step to becoming data literate. Whether you're making business decisions, verifying news, or just browsing online, you constantly work with data and information.

1.2 Data in the Digital Economy

The Digital Economy refers to economic activities that result from billions of online operations among people, businesses, devices, data, and processes. It is powered by digital technologies, especially the Internet, cloud computing, AI, and data. Figure 1.5 shows digital economy structure.

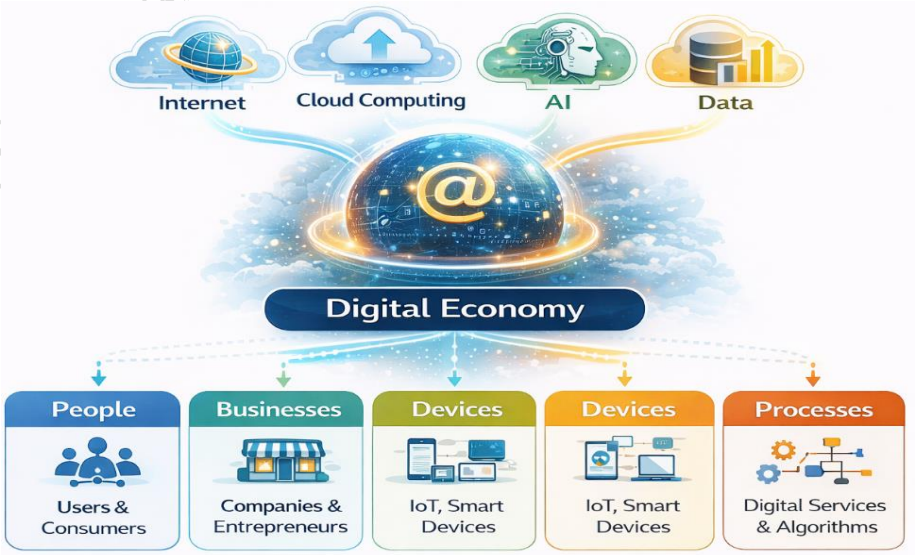


Fig. 1.5: Digital Economy Structure

Role of Data in the Digital Economy

Data is often called the "new oil" – a powerful, valuable resource. Just like oil powers industries, data powers the digital economy. Following are roles of data in the digital economy.

1. Data Drives Business Decisions

Various companies collect customer data through searches, clicks and purchases to understand preferences. For example, Amazon uses data to suggest products through recommendation systems.

2. Data Enables Innovation

Various startups use data to develop new services like personalized learning apps or smart health monitors. For example, Fintech apps use data to offer instant credit or insurance to users.

3. Data Creates New Jobs and Markets

Roles like data analyst, AI engineer, digital marketing expert, and Cybersecurity analyst are in high demand. These roles are created by data.

Online platforms like Uber, Ola, Swiggy and Zomato use real-time data for various operations. Real-time data is information that is collected and processed instantly, as events happen. It is immediate and constantly updated. It is used for location tracking, matching and for dynamic pricing / surge pricing. Figure 1.6 shows the data cycle.

Fig. 1.6: Data Cycle

4. Data Supports Governments and Public Services

Governments use data to deliver better services like smart city planning and digital health records. For example, the Aarogya Setu app in India used data to trace COVID-19 cases.

5. Data-Driven Advertising

Digital platforms such as Google and Facebook use targeted ads based on user data. This personalized advertising fuels major revenues in the digital economy.

Challenges in Digital Economy

There are many challenges in the digital economy. Some of them are given below.

1. *Privacy concerns:* Misuse of personal data
2. *Digital divide:* Unequal access to digital resources

3. *Data monopolies:* Few companies controlling massive data sets

In the digital economy, data is a core asset. Businesses, governments, and individuals must understand, manage, and protect data responsibly to drive growth, create innovation, and ensure inclusive development.

1.3 Ethical Considerations and Bias in Data Use

As data becomes central to decision-making in business, government, and society, it's important to use it responsibly and fairly. Two key issues are:

1. Ethical use of data
2. Bias in data and algorithms

1. Ethical Considerations in Data Use

Ethics in data means using data in a way that respects privacy, fairness, and transparency.

Key Ethical Principles:

Principle	Description
Privacy	Users' personal data must be collected and used with consent.
Transparency	People should know how their data is being used and by whom.
Security	Data must be protected from unauthorized access or leaks.
Accountability	Organizations must be responsible for the impact of data-driven decisions.

Example:

A health app that shares user data with third-party advertisers without user permission is violating ethical standards. Imagine you're using a health tracking app to monitor your fitness, heart rate, or menstrual cycle. You enter personal and sensitive data like: Weight and height, medical conditions, Sleep patterns, Mental health status and Daily habits.

You trust that the app will keep your data private and secure.

What the App Does Wrong:

Behind the scenes, the app sells or shares this data with third-party advertisers or companies — without informing or asking you.

An advertising company now knows your health condition or exercise patterns. You start seeing targeted ads for medicines or insurance based on that data. You never gave permission for this data to be shared. Figure 1.7 shows data privacy violations.

Fig. 1.7: Data Privacy Violation

2. Bias in Data and Algorithms

When we use AI in any application, then we need to train AI models by using data. Bias occurs when the data used to train AI or make decisions reflects prejudice, inequality, or stereotypes, often unintentionally.

Common Types of Bias:

Type	Description
Sampling Bias	Data is collected from a non-representative group
Labeling Bias	Human errors or stereotypes in how data is categorized
Algorithmic Bias	The model reflects the biased data it's trained on.

A facial recognition system trained mostly on lighter-skinned faces may perform poorly on darker-skinned individuals, leading to unfair outcomes.

Example of Data Bias:

A hiring algorithm used by a tech company preferred male candidates because it was trained on past data where most hires were men. This reinforced gender discrimination, even though the company didn't intend it.

How to Reduce Bias

Following measures can reduce bias and ensure ethical data use. Figure 1.8 shows bias reducing measures.

1. Use diverse and inclusive datasets
2. Audit and test algorithms regularly
3. Build explainable and transparent AI models
4. Ensure human oversight in critical decisions such as hiring and lending.

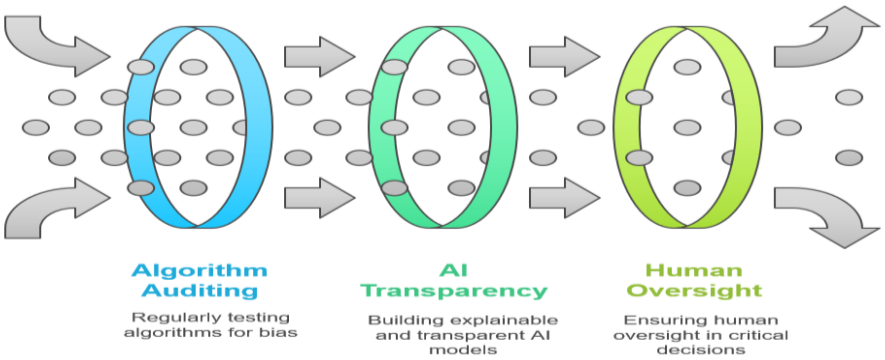


Fig. 1.8: Reducing Bias

While data has immense power, it must be used ethically and fairly. Ignoring ethical considerations and bias can lead to harmful, unjust, or discriminatory outcomes. Building trust and fairness in the digital world starts with responsible data practices.

Practical Activities

Activity 1.1. Spot the Fake Post (Data Literacy Exercise)

Task: Give students 3–4 viral social media posts such as, realistic but fictional. Some should be misleading such as, miracle cures, exaggerated claims, and some should be backed by reliable sources.

Steps

Step 1. Students question the source.

Step 2. Check reliability (WHO, Govt. portals, Research papers).

Step 3. Decide whether to share/report/ignore.

Outcome:

Students practice Riya's data literacy process to avoid misinformation.

Activity 1.2. Data vs Information Sorting Game

Task: Provide a list of items like: "Red", "25", "Mumbai", "Ravi scored 98 marks in Mathematics and lives in Mumbai."

Steps

Step 1. Students classify each item as Data (raw facts) or Information (processed with meaning).

Outcome: Learners clearly understand the difference between data and information.

Activity 1.3. Digital Economy Role-Play

Task: Divide the class into groups: Amazon, Uber, Government, Startup, Advertising Company.

Steps

Step 1. Each group explains how they collect and use data such as, Amazon uses purchase history for recommendations.

Step 2. Discuss benefits such as convenience, innovation vs. challenges such as privacy, monopolies.

Outcome: Students connect real-world industries to data use in the digital economy.

Activity 4: Data Privacy Debate

Scenario: "A health app shares user's fitness and medical data with advertisers without consent."

Task: Divide into For such as, app developers, advertisers and Against such as, users, privacy advocates.

Steps

Step 1. Debate on ethics, privacy, transparency.

Step 2. Suggest how the app can become *ethical and transparent.*

Outcome: Students apply ethical principles of data use such as, privacy, security, accountability.

Activity 1.5. Bias in Data Simulation

Task: Show students a small hiring dataset where most successful candidates are male.

Steps

Step 1. Ask them to train a mock AI using simple tables, not actual coding.

Step 2. Show how the system "prefers" male candidates.

Step 3. Then correct it by adding diverse and balanced data.

Outcome: Students see how sampling bias reinforces discrimination and how inclusive datasets reduce bias.

Activity 1.6. Data in Daily Life Diary

Task: Ask students to track how they generate or use data in a day (e.g., Google search, online shopping, using GPS, Instagram likes).

Steps

Step 1. Note down 5 activities.

Step 2. Identify: Who collects this data? How is it used?

Outcome: Students realize how deeply data is embedded in daily life and why literacy matters.

Summary

- Introduces data vs. information and their importance.
- Explains how data powers the digital economy in areas like business, innovation, jobs, government, and advertising.
- Highlights challenges: privacy, digital divide, monopolies.
- Covers ethics (privacy, transparency, security, accountability) and bias in data/AI (sampling, labeling, algorithmic).
- Emphasizes responsible use of data to avoid misinformation and unfair outcomes.

Check Your Progress

A. Multiple Choice Questions

1. Data becomes information when: (a) It is raw and unprocessed (b) It is organized and given meaning (c) It is stored in a database (d) It is collected from surveys

2. Which of the following is qualitative data? (a) 25 kg (b) 98 marks (c) Blue (d) ₹500
3. Real-time data is used by: (a) Amazon recommendation system (b) Uber surge pricing (c) WHO health reports (d) A textbook
4. An app secretly selling user health data violates which ethical principle? (a) Transparency (b) Monopoly (c) Privacy (d) Consistency
5. A hiring AI that prefers male candidates shows: (a) Algorithmic bias (b) Sampling bias (c) Typographical error (d) Data cleaning

B. Fill in the Blanks

1. _____ refers to raw facts, while _____ refers to processed data with meaning.
2. The digital economy is powered by technologies like the Internet, AI, and _____ computing.
3. Personalized online ads are an example of _____-driven advertising.
4. _____ occurs when AI reflects stereotypes or prejudice from its training data.
5. Respecting privacy, transparency, and accountability are part of _____ data use.

C. True or False

1. Data without context is not directly useful.
2. WHO reports are considered unreliable sources of data.
3. Information is always derived from data.
4. Data monopolies mean everyone has equal access to data.
5. Algorithmic bias can occur unintentionally.

D. Short Answer Questions

1. Differentiate between data and information with an example.
2. Why is data called the “new oil”?
3. List two challenges in the digital economy.
4. What are two key ethical principles in data use?
5. Give one example of bias in AI.

Session 2. Types and Sources of Data

Suppose a school decides to develop a Smart School Project – “Healthy Campus AI”. The objective of the project is to improve the health and learning environment in a school using data. While collecting the data students note that they need to collect different types of data by using different sources as given below:

1. Student conduct survey to collect data
2. Students rate their lunch 1 to 5 stars
3. Students describe how they feel after meals

4. Some data is stored in a spreadsheet
5. CCTV and Smart Cameras data
6. Used to observe student behavior in corridors
7. Facial expressions analyzed using AI
8. Smart Wristbands data
9. Measure students' physical activity data
10. Used to decide whether games will be indoor or outdoor

Students observe that in real life the data can be a variety of types. When they analyzed data, then the school changed its menu and gave students more breaks on hot days. Classes were adjusted based on activity levels from wristbands. Ultimately the school has happier, healthier students and better class performance. In this session we are going to discuss types and sources of data. Figure 2.1 shows Data Collection Methods.

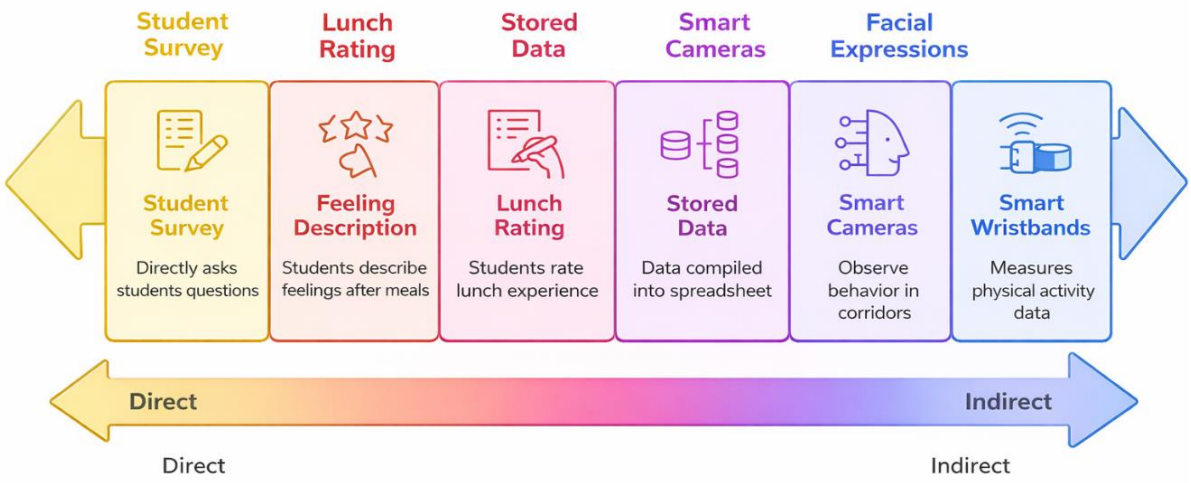


Fig 2.1: Data Collection Methods

2.1 Structured vs. Unstructured Data

Basically, there are two types of data such as:

1. Structured Data
2. Unstructured Data

Structured Data: Structured data that fits neatly into tables, rows, and columns. Easily stored, searched, and analyzed using software like Excel or SQL databases. It is best for numerical or clearly labeled information. Structured data is organized in rows and columns and It is easily stored in tables and databases. Excel sheets, SQL databases and employee records are examples of structured data. Figure 2.2 shows the structured data.

Other Examples:

1. Student records with Name, Roll No., Marks
2. Bank transaction logs
3. Inventory lists

Characteristic	Description
Format	Tables, rows, columns
Storage	Tables and databases
Software	Excel, SQL databases
Best Use	Numerical, labeled information
Examples	Student records, bank logs
Examples	Student records, bank logs

Fig. 2.2: Structured Data

Unstructured Data: This data does not follow a specific format. It is harder to organize and analyse. It usually comes from text, images, audio and video. It needs special tools such as AI and machine learning to process.

Structured data is like items neatly placed on a shelf. Unstructured data is like a box full of random objects. Not organized in a pre-defined format. It is difficult to analyse without processing.

Examples of unstructured data:

a) WhatsApp messages
b) Instagram photos
c) YouTube videos
d) Emails

A list of names and phone numbers is structured data and a folder full of selfies and voice notes is unstructured data. Figure 2.3 shows comparison of structured and unstructured data.

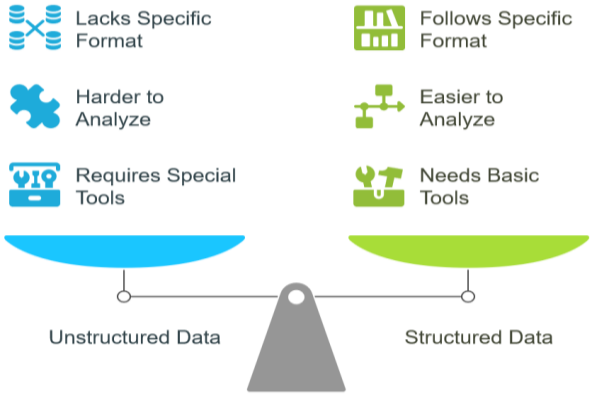


Fig. 2.3: Comparison of Structured and Unstructured Data

2.2 Quantitative and Qualitative Data

Quantitative data is numerical data that can be measured. The measurement such as, how much, how many, how often is a quantitative measurement. It is used in mathematics, statistics and surveys. If you can measure it, then it is quantitative. For example, age, height, income, and temperature are quantitative data. Figure 4 shows quantitative data uses.

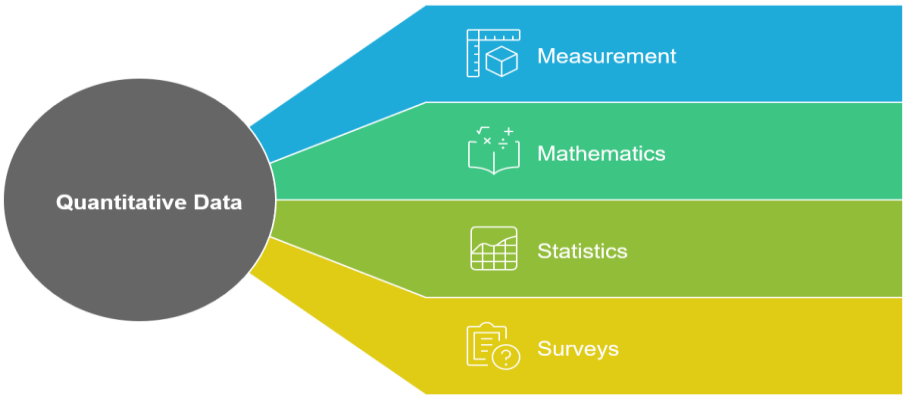


Fig. 2.4: Quantitative Data Uses

Examples of quantitative data:

1. 5 kilometers
2. 100 students
3. ₹10,000 income
4. 98% exam score

Qualitative data is descriptive data about characteristics. For example, eye color, opinion and brand preference is qualitative data. It describes qualities, characteristics and opinions. It is used in social sciences, interviews and reviews. If you can describe or observe it then it's qualitative. Figure 2.5 shows understanding qualitative data.

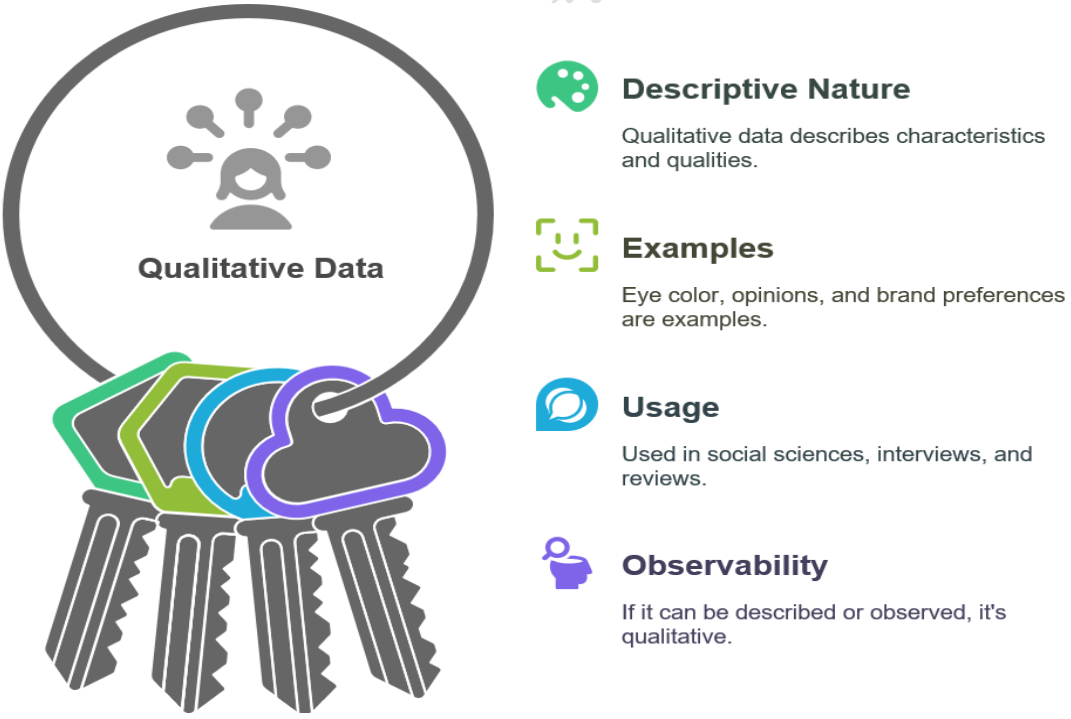


Fig. 2.5: Understanding Qualitative Data

Examples of qualitative data:

1. The shirt is "blue"
2. The food was "delicious"

3. Feedback: “Helpful and kind”

Uses of Qualitative and Quantitative Data in Computers

Computers process both quantitative (numerical) and qualitative (descriptive) data for various applications across fields like business, healthcare, education, artificial intelligence, and more. Here's how both types of data are used:

Uses of Quantitative Data in Computers:

1. *Data Analysis and Statistics:* Computers run statistical programs like Excel, R and Python to analyze sales numbers, exam scores, or performance metrics.
2. *Sensor Data Processing:* In smart devices such as fitness bands, weather sensors, computers collect and process numerical data like heart rate, temperature, and speed.
3. *Machine Learning Models:* Algorithms use numerical data such as, age, salary, click rates for training predictive models in AI and ML.
4. *Finance and Accounting Systems:* Computers process large volumes of quantitative financial data for budgeting, auditing, and forecasting.
5. *Simulations and Modeling:* Simulators use quantitative inputs to mimic real-world systems such as flight simulators and stock market models.

Uses of Qualitative Data in Computers:

1. *Text Processing and NLP (Natural Language Processing):* Computers analyze user reviews, feedback, and documents to identify sentiment, themes, and meaning.
2. *Image, Audio and Video Analysis:* AI systems interpret qualitative content like facial expressions, spoken language, or visual scenes.
3. *Chatbots and Virtual Assistants:* They understand and respond to human language using qualitative inputs such as, “How’s the weather?”.
4. *Decision-Making Systems:* Expert systems use qualitative rules such as, “If customer is unhappy, offer a discount” to simulate human decision-making.
5. *Content Categorization:* Computers categorize emails as spam or not, articles into topics, or customer complaints by type.

The following table shows a comparison of quantitative and qualitative data.

Aspect	**Quantitative Data**	**Qualitative Data**
Type	Numerical (e.g., 10, 98.6, 25%)	Descriptive (e.g., "happy", "low", "red")
Tools Used	Excel, R, Python, SQL	NLP tools, AI models, search algorithms
Application Examples	Finance, ML training, simulations	Chatbots, sentiment analysis, media tools
Output	Graphs, charts, statistical reports	Summaries, categories, insights

Numerical, Categorical, Ordinal, Textual Data

Numerical data is data that represents numbers which can be measured such as, 15, 23.4, 1000. Numerical data consist of pure numbers, either whole (discrete) or decimals (continuous) such as, 5 students, 12.5 kg.

Categorical data is data that is divided into groups or categories that have no numeric meaning. For example, Gender (Male/Female), Color Red/Green, City names is a categorical data. It belongs to categories or groups with no order.

Ordinal data is an ordered category where ranking matters, but not the difference between them. Ordinal Categories with a specific order such as, Small, Medium, Large or 1st, 2nd, 3rd, ratings: Poor, Average, Good, Excellent. Textual data consist of words, sentences, or paragraphs of text. Textual data contain full sentences, paragraphs, or comments where there is freeform text. For example, Reviews, comments, and tweets contain textual data. A product review: "Good quality, fast delivery" contains textual data. Figure 2.6 shows these data types.

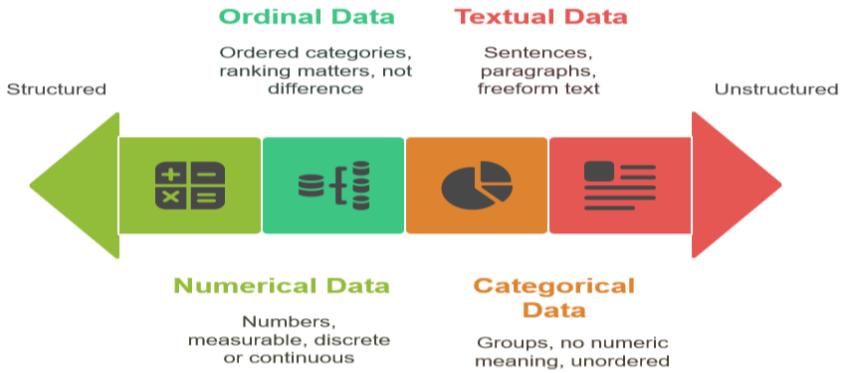


Fig. 2.6: Data Types

2.3 Data Collection Methods

Data collection is the process of gathering and measuring information on variables of interest, in a systematic way, to answer questions, test hypotheses, or build AI systems. In simple terms, data collection means collecting facts, figures, or opinions from various sources to help make decisions or create technology. Figure 2.7 shows the data collection process.

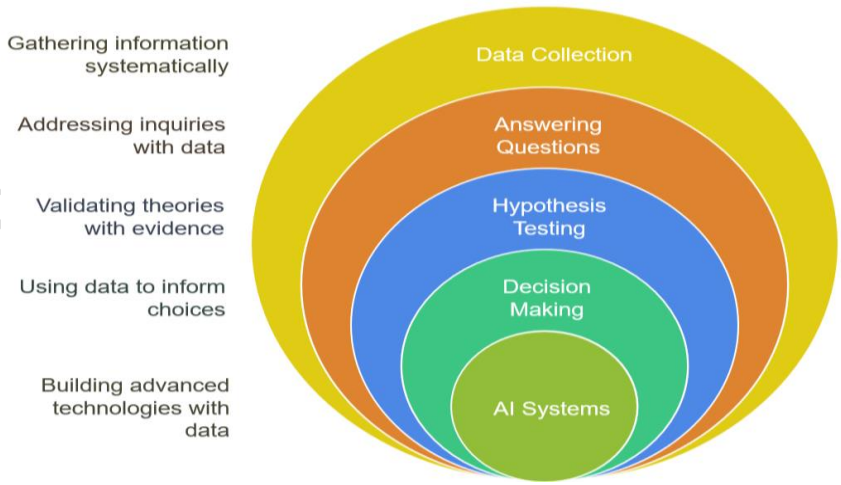


Fig. 2.7: Data Collection Process

Types of Data Collection Methods

Data can be collected in different ways, depending on the purpose, type of data (quantitative or qualitative), and available resources. Here are the main data collection methods:

1. Surveys and Questionnaires: A set of questions asked to people to collect information. Format can be online such as Google Forms, printed, or in person. It is used to collect opinions, preferences and feedback. It is useful in education, marketing and elections. It can be online or paper-based.

Types of Questions can be closed-ended such as, MCQs, Yes/No, Ratings for quantitative data. It can be open-ended such as, opinions, feedback for qualitative data. For example, a school asks students to rate their online class experience.

2. Interviews: A one-on-one conversation where the interviewer asks questions directly to the respondent. Types of interviews can be structured, that is, predefined questions or unstructured, that is, open discussion. For example, a teacher interviews students to know how they feel about AI in the classroom.

3. Observation: Watching and recording behaviors or events without direct interaction is called observation. It is often used when people's actions are more useful than their words. For example, a teacher observes how students use a new learning app in class. It is used in classrooms, wildlife studies and customer behavior. For example, noting how many students raise hands in class when a question is asked.

4. Sensors and Devices (Automated Collection): Devices and sensors automatically record data without human input. It is used in Smart devices, IoT, fitness bands and vehicles. For example, a smartwatch collects data on steps, heart rate, and sleep patterns.

5. Web Scraping / Online Sources: Collecting data from websites, social media, blogs, etc., using software tools. It is used in market research, news monitoring and sentiment analysis. For example, an AI tool collects tweets to see public opinion on a new movie.

6. Experiments: Data is collected by conducting experiments under controlled conditions. It is used mostly in scientific research and product testing. For example, a company tests two versions of a website to see which design users prefer. Figure 2.8 shows data collection methods.

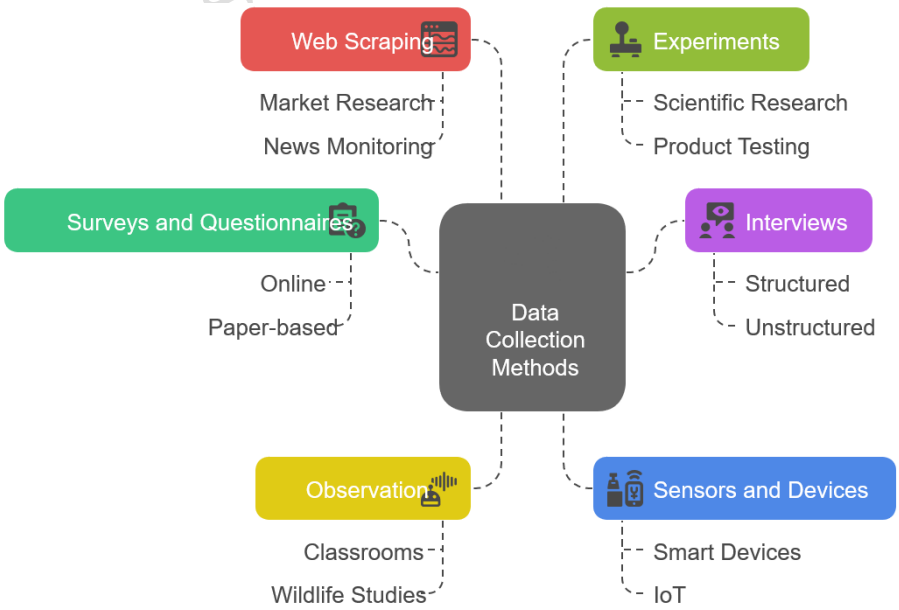


Fig. 2.8: Data Collection Methods

Choosing the Right Method: Choosing the right method depends upon the situation as given below.

Situation	Recommended Method
Want numbers/statistics	Surveys, sensors
Want opinions or feelings	Interviews, open surveys
Want to watch behavior	Observation
Want online trends	Web scraping
Want to test a hypothesis	Experiments

Advantages and Disadvantages: Advantages and disadvantages of data collection methods are as given below.

Method	Advantages	Disadvantages
Surveys	Quick, reach many people	May get incomplete or false answers
Interviews	In-depth information	Time-consuming
Observation	Real behavior data	Observer bias, time-consuming
Sensors	Real-time, accurate	Expensive, needs tech setup
Web Scraping	Large, online data access	May face legal or ethical issues
Experiments	Scientific, controlled results	Costly, hard to replicate sometimes

Why Is Data Collection Important in AI?

Following are reasons for the importance of data collection in AI.

1. AI systems need huge amounts of data to learn and make smart decisions.
2. Without proper data collection, AI models may become biased or inaccurate. *Example:* A facial recognition AI must be trained on diverse face data to work well for everyone.
3. Data collection is the first and most important step in any data or AI project.
4. The method chosen should match the goal of the study or system.
5. Both manual and automated methods are important in today's digital world.

Practical Assignment

Project Idea: "Survey Your Classmates"

Create a 5-question survey about smartphone usage. Collect answers from 20 classmates. Create a small report with charts showing the results. Discuss: Which method did you use and why?

2.4 Open Data and Big Data Concepts

Open Data is data that is freely available to everyone to use, share, and reuse. It promotes transparency, research, and innovation. For example, Government crime statistics, public weather data is open data. Open data that is publicly available, free to use, modify, and share. It is shared by governments, organizations, and researchers to promote transparency.

Examples of open data:

a) Indian government census data
b) WHO COVID-19 statistics
c) City air quality data

Use Case: A startup uses open data to develop an app that warns about air pollution in real-time.

Big Data is a massive volume of data generated rapidly from various sources. It refers to huge, complex datasets that grow rapidly and need advanced tools such as AI and cloud computing to analyze it.

It has following characteristics (5Vs):

- *Volume* – Huge amount of data. For example, Facebook generates TBs of data daily.
- *Velocity* – Data is created very fast. For example, the Stock market updates every second.
- *Variety* – Comes in many forms such as, text, image, video, GPS data and logs.
- *Veracity* – Data quality and accuracy is varied. Data Veracity refers to the accuracy, reliability, and trustworthiness of the data. Not all data is correct or useful.
- *Value* – It is possible to extract useful insights from it such as, predicting customer churn from data patterns.

For example, social media platforms analyze millions of daily posts (Big Data).

Use Case: Amazon analyzes millions of transactions (Big Data) to suggest what you may want to buy next. Figure 2.9 shows comparison between open data and big data.

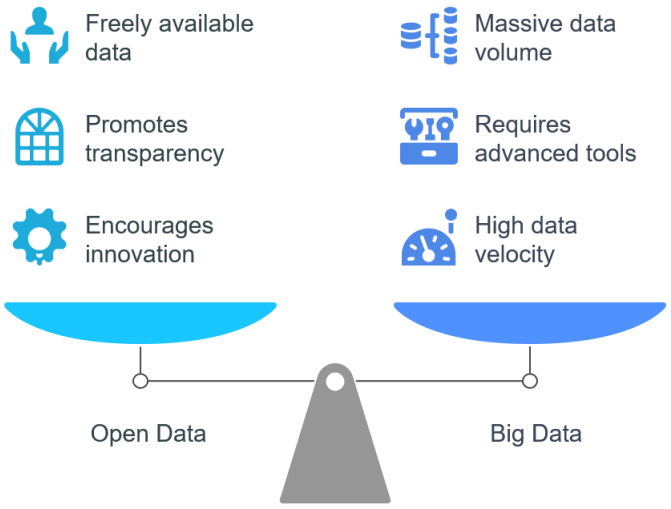


Fig. 2.9: Comparison between open data and big data

Practical Activities

Activity 2.1. Structured vs. Unstructured Sorting

Task: Provide students with a mixed list of items:

"Riya, Roll No. 21, 98 Marks", "Instagram photo", "Excel Sheet", "WhatsApp Voice Note".

Steps

Step 1. Students classify each as Structured (fits into rows & columns) or Unstructured

(images, text, audio).

Step 2. Discuss why structured is easier for Excel/SQL, while unstructured needs AI tools.

Outcome: Learners see how school spreadsheets differ from selfies and chats.

Activity 2.2. Quantitative vs. Qualitative in Healthy Campus AI

Task: Use examples from the project:

"Lunch rated 4 stars" → Quantitative

"Food was delicious" → Qualitative

Steps

Step 1. Students collect 5 examples each from their daily life (e.g., "Temperature 32°C" vs. "Weather is pleasant").

Step 2. Place them under Quantitative or Qualitative.

Outcome: Students learn measurement vs. description in data.

Activity 2.3. Numerical, Categorical, Ordinal, Textual Game

Task: Give dataset samples:

(i) Numerical: "Steps = 5,000"

(ii) Categorical: "City = Mumbai"

(iii) Ordinal: "Rating = Poor, Average, Good"

(iv) Textual: "Comment = 'I feel tired after lunch'"

Steps

Step 1. In groups, students label each correctly.

Outcome: Students practice breaking down complex datasets into subtypes.

Activity 2.4. Data Collection Role-Play

Task: Divide students into 5 groups, each act as one data collection method:

1. Surveys (prepare 3 questions),
2. Interviews (ask classmates how they feel after meals),
3. Observation (note who uses playground),
4. Sensors (simulate step counts with mock data),
5. Web scraping (pretend to check online school reviews).

Steps

Step 1. Each group collects small data from class.

Step 2. Present which type of data they got (quantitative or qualitative, structured or unstructured).

Outcome: Students experience data collection hands-on.

Activity 2.5. Mini Healthy Campus Project

Task: Create a small dataset (10 students):

1. Lunch rating (1–5) → Quantitative, Numerical

2. Comment about lunch → Qualitative, Textual
3. Steps walked (from wristband) → Structured, Numerical
4. CCTV note on corridor behavior → Unstructured, Observation

Steps

Step 1. Students enter structured data into Excel.

Step 2. Write down qualitative feedback.

Step 3. Discuss how a school could use this to change menu, games, and classes.

Outcome: A practical simulation of how schools use real data.

Activity 2.6. Open Data Treasure Hunt

Task: Ask students to find 1 open dataset online (Govt. census, WHO COVID stats, weather data).

Steps

Step 1. Identify whether it's open data or not.

Step 2. Present one insight (e.g., "WHO shows daily vaccination rates").

Outcome: Students realize public data sources are powerful for innovation.

Activity 2.7. Big Data in Everyday Life Discussion

Task: Give examples like:

Facebook daily posts, Amazon purchases, YouTube recommendations.

Steps

Step 1. Students identify Volume, Velocity, Variety, Veracity, Value for each.

Outcome: Students connect Big Data's 5Vs with real platforms they use daily.

Summary

1. Explains structured vs. unstructured data (organized tables vs. free-form like images, videos).
2. Covers quantitative vs. qualitative data and their uses in computers.
3. Introduces numerical, categorical, ordinal, and textual data.
4. Discusses data collection methods: surveys, interviews, observation, sensors, web scraping, experiments.
5. Explains open data (free for public use) and big data (large, complex datasets with 5Vs: Volume, Velocity, Variety, Veracity, Value).
6. Shows how schools or projects can apply these concepts in real life.

Check Your Progress

A. Multiple Choice Question

1. Which is an example of structured data? (a) Instagram photo (b) YouTube video (c) Bank transaction log (d) WhatsApp message

2. Which data type describes qualities like color or mood? (a) Quantitative (b) Qualitative (c) Ordinal (d) Numerical
3. "Small, Medium, Large" is an example of: (a) Numerical data (b) Ordinal data (c) Textual data (d) Categorical data
4. Which is NOT a data collection method? (a) Survey (b) Web scraping (c) Observation (d) Guessing
5. The "5Vs" of big data do NOT include: (a) Velocity (b) Variety (c) Value (d) Visualization

B. Fill in the Blanks

1. _____ data fits into rows and columns, while _____ data is messy and unorganized.
2. "Eye color: Brown" is an example of _____ data.
3. Reviews and comments are examples of _____ data.
4. _____ data is freely available for public use.
5. Fitness bands and IoT devices collect data using _____.

C. True or False

1. Structured data is easier to analyze than unstructured data.
2. Quantitative data cannot be measured.
3. Ordinal data has order but no exact difference between ranks.
4. Big data is small, simple datasets.
5. Open data promotes transparency and innovation.

D. Short Answer

1. Explain structured vs. unstructured data with examples.
2. Give one example each of quantitative and qualitative data.
3. What are the 5Vs of Big Data?
4. List three methods of data collection.
5. What is open data? Give an example.

Session 3. Data Quality and Data Cleaning

Imagine a school is preparing the final results of Class XI students for a board inspection. The data is entered into a computer system by multiple teachers. After examining the data, they observe that some marks are missing, some students got more than the maximum valid score, the total of marks is incorrect or there are blank totals. If such data is processed then we may generate inaccurate results. So, we observe that real-world data is often messy. Computers don't "understand" wrong data unless we clean it. Accurate, complete, and consistent data is the foundation of good decisions and

smart AI. In this session we are going to discuss data quality and data cleaning. Figure 3.1 shows inaccurate results due to messy data.

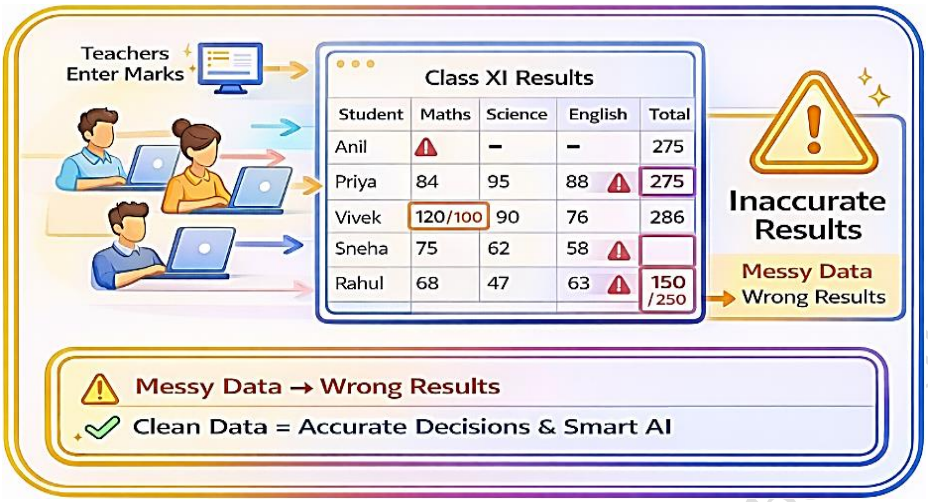


Fig. 3.1: Inaccurate Results due to messy data

Data Quality and Cleaning

Data Quality means how suitable the data is for analysis, decision-making, or use in computer systems like AI. Poor quality data can lead to wrong results.

Dimensions of Data Quality

The three most important dimensions of data quality are:

A. Accuracy: Accuracy means how correct the data is. It should represent the real-world values correctly. For example, if a student scored 85 marks, but the system shows 58, then, it's inaccurate. Incorrect data gives incorrect results in reports, AI models, and decisions.

B. Completeness: Completeness means whether all necessary data is available. Missing values reduce completeness. For example, a form without the student's email or a health app missing your weight info. Incomplete data leads to errors in analysis or AI training.

C. Consistency: Consistency means data is uniform across systems and entries. For example, a student's name written as "Rahul Sharma" in one record and "R. Sharma" in another – this is inconsistent. Consistent data helps avoid duplication and confusion. Figure 3.2 shows data quality dimensions.

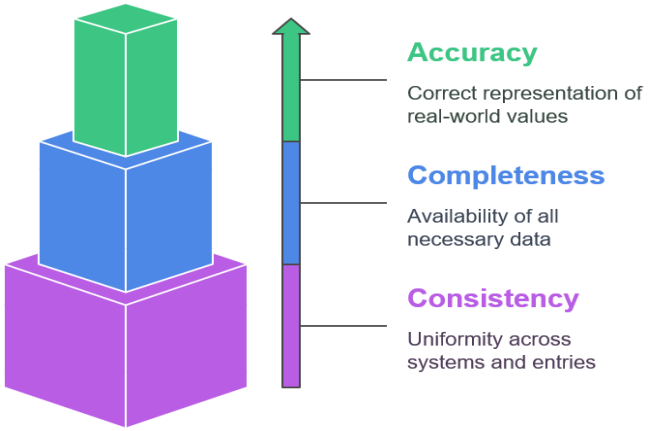


Fig. 3.2: Data Quality Dimensions

Data Cleaning Techniques

Data Cleaning is the process of fixing or removing incorrect, incomplete, or duplicated data to improve quality.

Common Data Cleaning Techniques:

1. *Removing Duplicates:* Deleting repeated entries of the same data.
2. *Filling Missing Values:* Replacing blank fields with average, most common value, or using prediction.
3. *Fixing Outliers:* Correcting or removing extreme values that don't match real-world logic.
4. *Standardization:* Making data formats uniform such as, date format, decimal places.
5. *Validation:* Checking if the data follows rules such as, age must be between 0–120. Figure 3.3 shows data cleaning techniques.

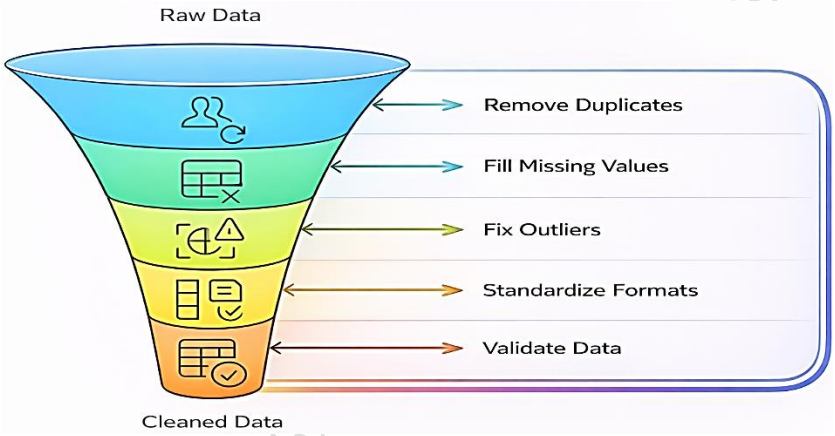


Fig. 3.3: Data Cleaning Techniques

Example:

If students' scores are like: [90, 95, NaN, 600, 85, 85]

Where NaN is a missing value, 600 is an outlier (score above max 100), Two 85s might be a duplicate. Cleaning will make the data reliable.

Common Data Errors

Here are typical errors that happen during data collection or entry:

1. ***Missing Values:*** In this type of error, data fields are left blank or lost. For example, no email is entered in a contact form.
2. ***Deletion Errors:*** In this error, important data is wrongly deleted. For example, deleting an entire row with valid information.
3. ***Outliers:*** Here, values that are extremely high/low and unrealistic. For example, a student scoring 500 out of 100.
4. ***Data Inconsistency:*** In this type of error, conflicting or non-uniform information across entries appears. For example, date format: DD/MM/YYYY vs MM/DD/YYYY.
5. ***Typographical Errors:*** Here mistakes appear during typing of data. For example, typing "Rahul" as "Rhaul".

6. ***Incorrect Data Types:*** In this type of error, text is entered where numbers are expected or vice versa. For example, writing "eighty" instead of 80. Figure 4 shows common data errors.

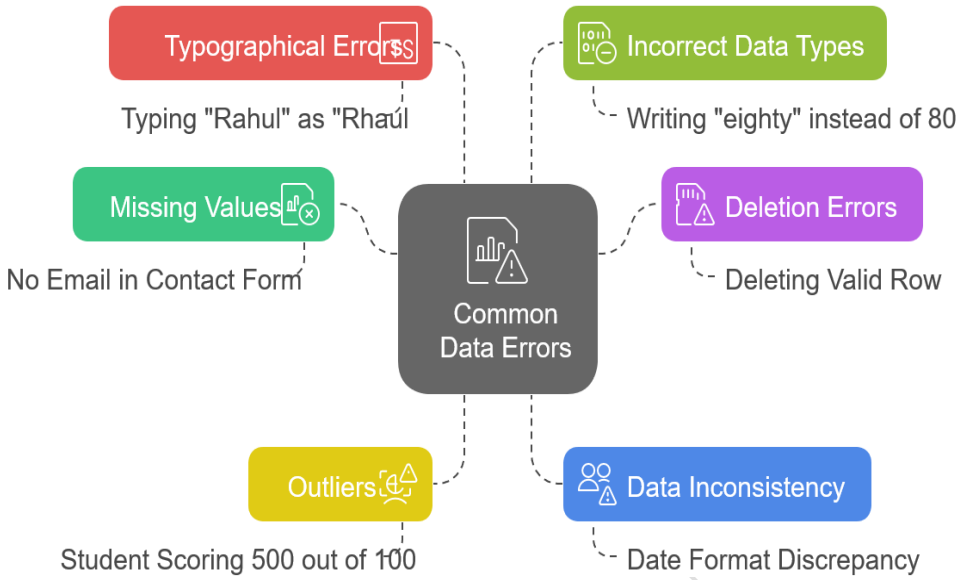


Fig. 3.4: Common Data Errors

Handling Common Errors in Data Cleaning

Errors in data can be handled by using the following solution.

1. *Missing Values:* Fill with average/median value, or use predictive models, or flag for review.
2. *Outliers:* Validate and correct, or remove if clearly wrong.
3. *Inconsistencies:* Use standard formats for dates, names, units and all other such entities.
4. *Duplicates:* Remove repeated rows or merge them together.
5. *Typing Errors:* Use spell-check tools or validation rules for correcting such errors.

Importance of Data Cleaning

1. Good data = Better decisions
2. Clean data = Better AI models
3. In real life, Banks, hospitals, schools, and governments rely on clean and high-quality data to function smoothly.
4. Data Quality = Accuracy + Completeness + Consistency.
5. Data Cleaning = Correcting errors to improve data quality.
6. Errors like missing values, outliers, and inconsistencies must be fixed before analysis. Figure 5 shows data quality hierarchy.

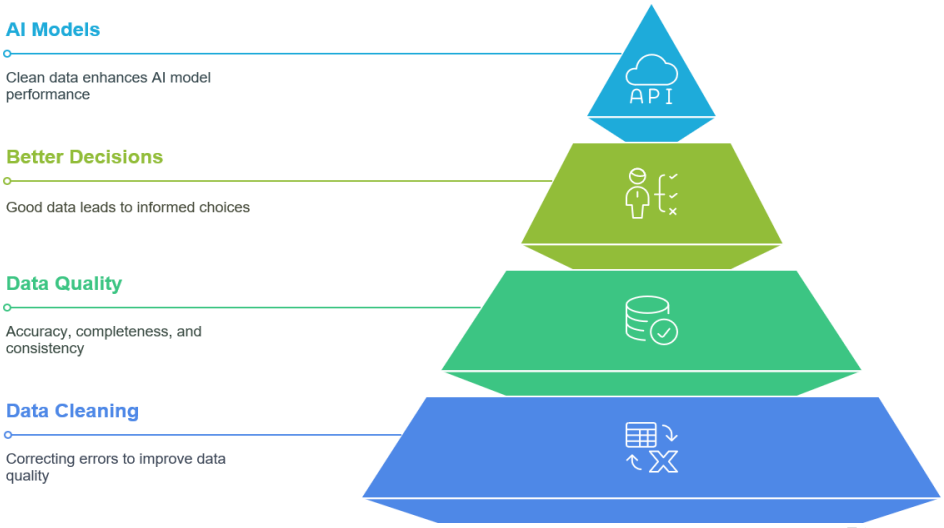


Fig. 3.5: Data Quality Hierarchy

Practical Activities

Activity 3.1

"Fix the Marks List": Give students a faulty table of exam marks with missing values, outliers, and duplicates. Ask them to clean and recalculate the totals.

Activity 3.2. Spot the Errors

Objective: Identify common data quality issues.

Steps:

Step 1. Provide students with a faulty marks list, e.g.

Roll No	Name	Maths	Science	English	Total
1	Rahul Sharma	95	88		150
2	R. Sharma	105	90	87	282
3	Priya Singh	78	NaN	85	
4	Aman Gupta	85	85	85	255
5	Aman Gupta	85	85	85	255
6	Reena Patel	60	65	70	190

Step 2. Ask students to circle errors such as:

1. Missing values (English marks blank, NaN in Science).
2. Outliers (Maths score = 105, above 100).
3. Inconsistent names (Rahul Sharma vs. R. Sharma).
4. Duplicate rows (Aman Gupta repeated).
5. Incorrect totals (Row 1, Row 3).

Step 3. Discuss: Why are these errors problematic?

Activity 3.3: Clean the Dataset

Objective: Apply data cleaning techniques.

Steps

Step 1. Using the above faulty table, ask students to:

1. Fill missing values (use average marks of that subject).
2. Correct outliers (limit to max 100).
3. Standardize names (use full names).
4. Remove duplicates (keep only one Aman Gupta).
5. Fix totals (recalculate correctly).

Step 2. Students prepare the corrected version and compare in groups.

Activity 3.4. Data Detective Roleplay

Objective: Develop problem-solving and critical thinking.

Steps

Step 1. Divide class into 3 groups:

1. Accuracy Detectives → Check correctness.
2. Completeness Detectives → Find missing data.
3. Consistency Detectives → Check uniformity of entries.

Step 2. Each group presents errors they found and solutions.

Activity 3.5. Real-Life Connection

Objective: Relate school data cleaning to real-world applications.

Task:

Ask students: *"What if banks stored your account balance incorrectly? What if a hospital entered wrong blood group data?"*

Students share examples where data cleaning is critical in real life.

Activity 3.6. Mini Project – Fix Your Own Data

Objective: Apply concepts in a personal dataset.

Task:

1. Students collect 10 rows of self-created data (like daily study hours, screen time, or steps walked).
2. Intentionally add errors (missing, duplicates, outliers).
3. Exchange with a classmate and clean each other's dataset.
4. Present cleaned vs. raw dataset.

Summary

1. Stresses that real-world data is often messy and needs cleaning.
2. Defines data quality dimensions: accuracy, completeness, consistency.
3. Introduces data cleaning techniques: removing duplicates, filling missing values, fixing outliers, standardizing formats, validation.

4. Lists common data errors: missing values, deletion errors, outliers, inconsistencies, typos, wrong data types.
5. Explains why clean data = better decisions and AI models.

Check Your Progress

A. Multiple Choice Questions

1. Which is NOT a dimension of data quality? (a) Accuracy (b) Completeness (c) Consistency (d) Randomness
2. A score of 600 out of 100 is an example of (a) Missing value (b) Outlier (c) Inconsistency (d) Typo
3. Which error occurs when a name is entered differently in two records? (a) Duplicate (b) Inconsistency (c) Missing value (d) Deletion
4. Which technique deals with repeated rows? (a) Filling missing values (b) Removing duplicates (c) Standardization (d) Validation
5. Typing "Rhaul" instead of "Rahul" is a (a) Data type error (b) Outlier (c) Typographical error (d) Inconsistency

B. Fill in the Blanks

1. Data quality depends on accuracy, completeness, and _____.
2. _____ values reduce the completeness of data.
3. _____ is the process of fixing incorrect, incomplete, or duplicated data.
4. Standardization ensures uniform _____ across datasets.
5. Clean data = better _____ and AI models.

C. True or False

1. Outliers always indicate wrong data.
2. Duplicate rows can lead to misleading results.
3. Validation ensures that data follows set rules.
4. Data errors cannot be fixed once recorded.
5. Banks and hospitals rely on clean data for smooth operations.

D. Short Answer Questions

1. Define data quality.
2. What are the three dimensions of data quality?
3. Give two examples of common data errors.
4. Why is data cleaning important in AI?
5. Mention two data cleaning techniques.

Session 4. Statistics for Data Analysis

The school canteen wants to improve its food quality. They give a feedback form to 30 students, asking them to rate the food taste on a scale of 1 to 10. Now they want to analyse the data to understand student feedback. To find central tendency of feedback they need to use statistical techniques such as mean, mode and variance. Range can be used to see variation in opinions. Visualization Principles are used to present ratings clearly. Inferential Statistics can be used to compare feedback between two classes using hypothesis testing. Figure 4.1 shows school canteen feedback analysis. In this session we are going to discuss these statistical techniques.

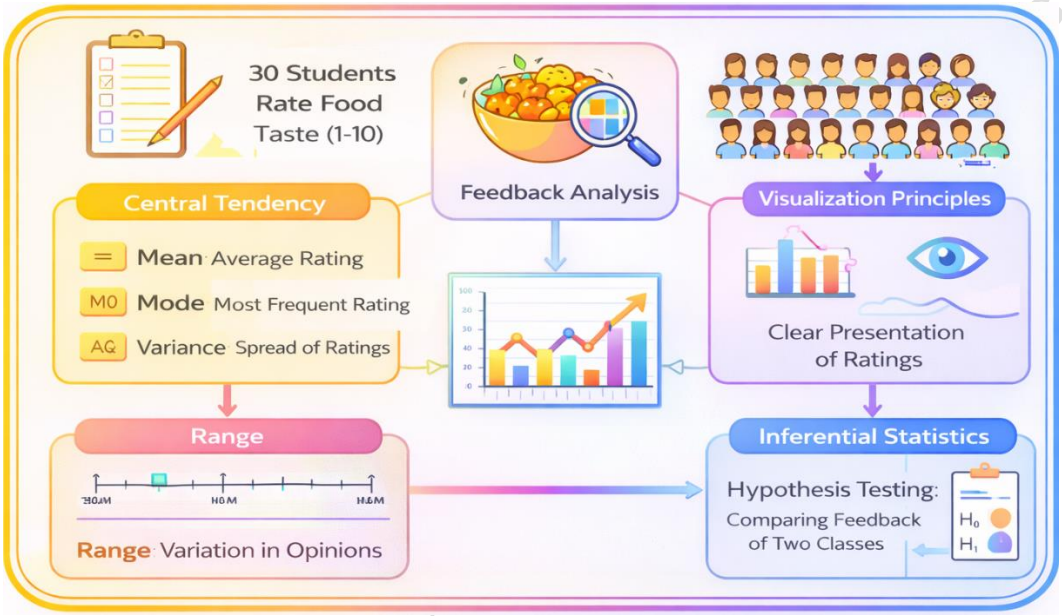


Fig. 4.1: School Canteen Feedback Analysis

4.1 Statistics in Data Science

This section helps us understand data and draw conclusions on data by using different methods.

Descriptive Statistics

Descriptive statistics is about summarizing and describing data using numbers. It helps us to get a quick idea of what the data is saying.

Mean (Average): The sum of all values divided by the number of values.

Formula:

Mean = Sum of all values/ Number of values

Example:

Marks = 70, 80, 90

Mean = (70 + 80 + 90) ÷ 3 = 80

Median (Middle Value): The middle value in an ordered dataset.

Steps:

1. Arrange data in ascending order.
2. If odd count, pick the middle value.

3. If even count, average the two middle values.

Example:

Data = 50, 60, 70 → Median = 60

Data = 40, 50, 60, 70 → Median = (50+60)/2 = 55

Mode (Most Frequent Value): The value that appears most often in the data.

Example:

Data = 5, 7, 7, 8, 9 → Mode = 7

Range (Spread of Data): The difference between the maximum and minimum values.

Formula:

Range = Maximum − Minimum

Example:

Data = 10, 20, 30 → Range = 30 − 10 = 20

Standard Deviation

Standard deviation (SD) is a measure of how spread out or scattered the values in a data set are around the mean (average).

1. A low SD means data points are close to the mean.
2. A high SD means data points are more spread out from the mean.

Formula (for sample SD):

$$s = \sqrt{\frac{1}{n-1}\sum(x_i - \bar{x})^2}$$

Where:

x_i = each value

$\bar{x}$ = mean

n = number of values

Example:

Suppose 5 students scored in a test:

Scores: 70, 72, 68, 74, 76

Step 1: Calculate the Mean

$\bar{x}$= 70+72+68+74+765 = 3605 = 72

Step 2: Subtract the mean from each score and square the result:

$(70 - 72)^2 = 4$

$(72 - 72)^2 = 0$

$(68 - 72)^2 = 16$

$(74 - 72)^2 = 4$

$(76 - 72)^2 = 16$

Step 3: Find the average of these squared differences:

$$\text{Variance} = \frac{4+0+16+4+16}{5-1} = \frac{40}{4} = 10$$

Step 4: Take the square root:

$$\text{Standard Deviation (s)} = \sqrt{10} \approx 3.16$$

Interpretation: On average, the scores deviate about 3.16 marks from the mean.

Importance of Standard Deviation

Use	Example
Understand data spread	Exam scores of two classes with same average
Compare consistency	Player performance in cricket or football
Risk analysis	Investment returns (finance)
Grading and academic analysis	Variance in student marks

Practical Activity

A teacher wants to understand average performance, common score, typical score unaffected by extremes, performance gap for a Mathematics class test. This can be achieved by using mean, median, mode, and range using students' test scores.

Step 1. Test Scores of Students

Let's say five students in a class scored the following marks in a math test:

Scores: 65, 70, 75, 70, 90

Step 2. Compute Mean (Average)

Definition: The sum of all values divided by the number of values.

Mean = (65+70+75+70+90) =3705/5=74

So, the average (mean) score is 74. Average performance of class is 74.

Step 3. Compute Median (Middle Value)

Definition: The middle value when numbers are arranged in order.

Ordered scores: 65, 70, 70, 75, 90

The middle (3rd) score is 70, so Median = 70. The common score of the class is 70.

Step 4. Mode (Most Frequent Value)

Definition: The value that appears most often.

In this case, 70 appears twice, more than any other score. Mode = 70. This is the typical score of the class.

Step 5. Range (Spread of Data)

Definition: Difference between the highest and lowest values.

Range = 90−65 = 25. Range = 25. This is a performance gap in the test.

1.1 Data Visualization Principles

Data Visualization means using charts, graphs, and images to make data easier to understand.

Key Principles:

1. Clarity: The chart should be easy to read.
2. Accuracy: Must reflect real data correctly.
3. Simplicity: Avoid too much decoration or 3D effects.
4. Labeling: All axes and data points should be labeled.
5. Appropriate Type: Use the correct type such as, bar, pie or line.

Common Graph Types:

1. *Bar Graph:* This graph is used to compare categories such as, bar graph of number of students in each class. A typical bar graph is shown in Figure 4.2.

Fig. 4.2: Bar Graph

2. *Pie Chart:* This graph is used to show percentage or part of a whole. A typical pie chart is shown in Figure 4.3.

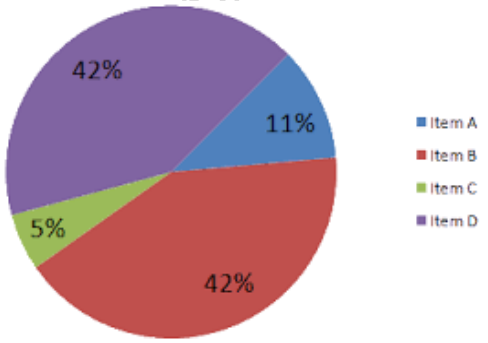


Fig 3: Pie Chart

3. *Line Graph:* This graph is used to show trends over time such as monthly sales. Figure 4.4 shows a typical line graph.

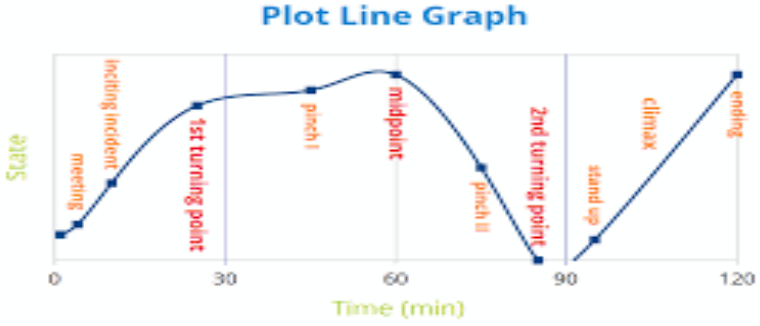


Fig. 4.4: Line Graph

4. *Histogram:* This graph is used to show frequency of data in ranges. Figure 4.5 shows a typical histogram.

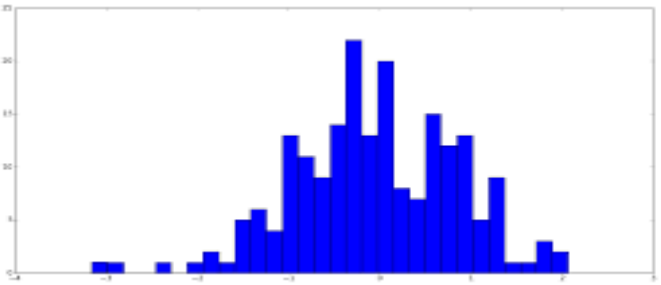


Fig. 4.5: Histogram

Good visualization helps us spot patterns and make decisions faster.

Practical Activity

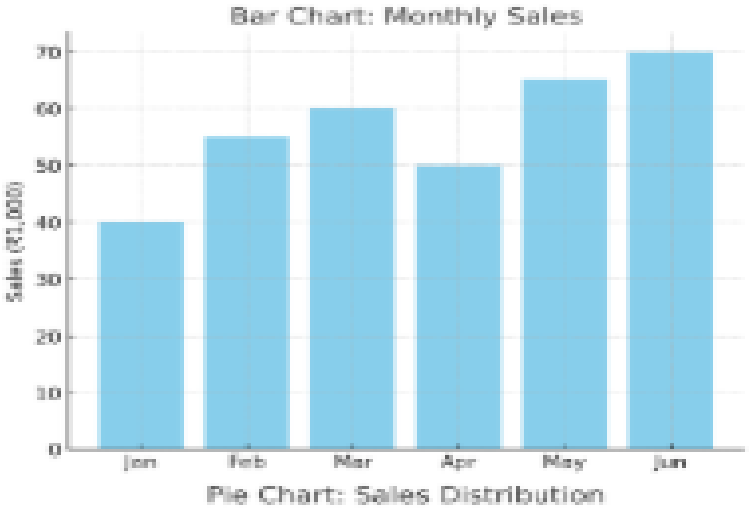
Here's a simple example shown using different types of data visualizations—bar chart, pie chart, line graph, and histogram—to help you understand how the same data can be visualized in various ways.

Consider a Monthly Sales Data of a Stationery Shop (in ₹1,000) as given in table.

Month	**Sales (₹)**
January	40
February	55
March	60
April	50
May	65
June	70

1. Bar Chart

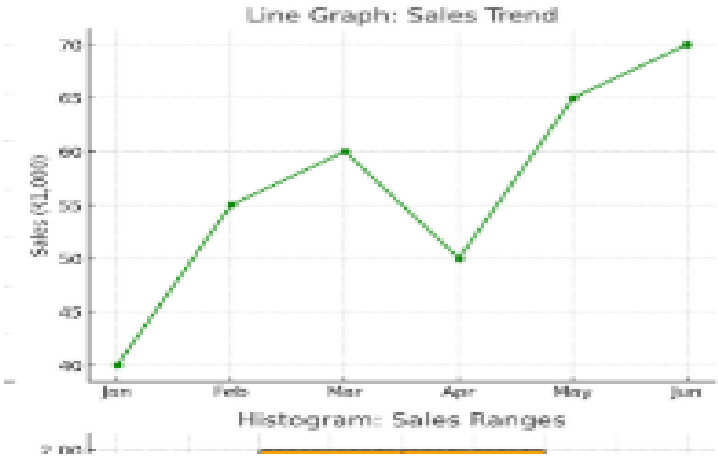
It shows sales for each month using rectangular bars.



Observe that it is easy to compare values month-to-month.

2. Line Graph

It shows trends over time by connecting data points with lines.



Observe that it is best for showing growth or decline over time.

3. Pie Chart

It shows each month's sales as a **proportion** of the total sales.

Let's say total sales = ₹340

January: 11.8%

February: 16.2%

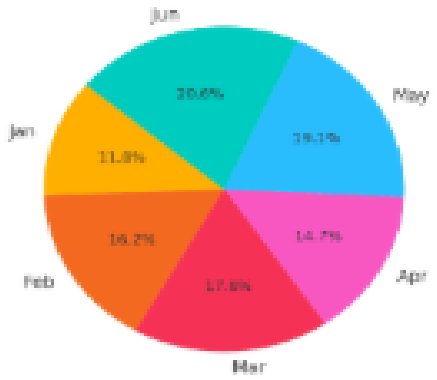
March: 17.6%

April: 14.7%

May: 19.1%

June: 20.6%

In a pie chart, each "slice" would represent a month's share of the yearly sales.



Observe that this graph is good to show percentage contribution or part of the whole.

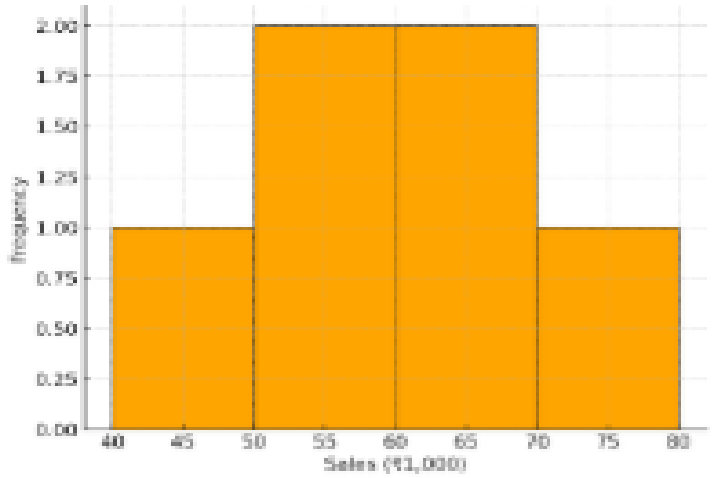
4. Histogram

Let's group the sales into ranges:

Sales Range (₹)	Frequency
40–49	1
50–59	2

Sales Range (₹)	Frequency
60–69	2
70–79	1

Now plot frequency vs sales range:



The graph shows distribution of values over intervals.

Choosing the Right Graph

Choosing the right graph depends on the type of data you have and what you want to show. For example, a bar graph is suitable for comparing individual values. The line graph shows trends over time. Pie chart shows part of a whole. Histogram shows distribution of data.

Graph Type	Best Use
Bar Chart	Comparing individual values
Line Graph	Showing trends over time
Pie Chart	Showing parts of a whole
Histogram	Showing distribution of data

1.2 Avoiding Misleading Visualizations

1. Start the Y-axis at Zero: Truncating the y-axis can exaggerate differences.

Example of Misleading Chart:

1. Bar chart showing sales where Y-axis starts at 50 instead of 0.
2. It makes ₹60K vs ₹65K look drastically different, though it's just ₹5K.

Always start axes at zero unless there's a valid statistical reason and it's clearly explained.

2. Maintain Proportions: Pie charts or bar sizes that don't reflect actual data proportion are misleading.

Misleading Pie Chart:

1. If slice sizes don't match the actual percentages.

Always use accurate scales and angles.

3. Avoid 3D Effects: 3D visuals can distort perception of size, especially in pie charts.

Example:

1. A 3D pie chart makes a smaller category look bigger because of the viewing angle.

Use flat, 2D charts for clarity and accuracy.

4. Choose the Right Chart Type: Wrong chart types can confuse or hide the real message.

Example:

1. Using a pie chart for a time series (monthly sales) is not suitable.

Use line graphs or bar charts for trends over time.

5. Avoid Cherry-Picking Data: Showing only selected data points to support a claim is misleading.

Example:

1. Showing only 3 months out of 12 to make growth seem constant.

Show complete and relevant data unless there's a good reason not to.

6. Be Consistent with Scale and Units: Mixing units or changing scales mid-graph confuses viewers.

Example:

1. One bar in ₹ and another in $ can be misleading comparisons.

Use a single unit and consistent scale.

7. Label Everything Clearly: Missing or vague labels make interpretation difficult.

Always include:

a) Titles
b) Axis labels
c) Legends
d) Units

Table

Mistake	**Why It's Misleading**	**Fix**
Y-axis not starting at 0	Exaggerates small differences	Start Y-axis at 0
Distorted proportions	Misrepresents data magnitude	Use proper scaling
3D Effects	Distorts perception	Use 2D visuals
Wrong chart type	Hides real patterns	Use correct chart for data
Cherry-picked data	Gives false impression	Show complete data
Inconsistent units/scales	Makes comparisons invalid	Standardize units and scales
No labels/titles	Confuses viewer	Add clear labels and legends

Real-World Example:

A news channel shows a bar graph of job growth where the Y-axis starts at 95, making a small increase look huge. Viewers may wrongly believe there's a job boom. Show full scale, include context, and label axes.

1.3 Data Storytelling – Making Data Speak with Meaning

Data storytelling is the art of combining data, visuals, and narrative to communicate insights clearly, effectively, and memorably.

Data storytelling = Data + Visuals + Human Narrative

It's not just showing data, but helping people see what it means and why it matters. It's not just about charts and numbers — it's about turning raw data into a story that informs, persuades, or inspires action.

Key Elements of Data Storytelling

Element	Role	Example
Data	The facts, numbers, and evidence.	Sales figures, test scores, survey results.
Visuals	Graphs and charts that make data easier to grasp.	Bar charts, maps, infographics.
Narrative	The human context that explains "why it matters."	"Sales dropped due to lockdowns."

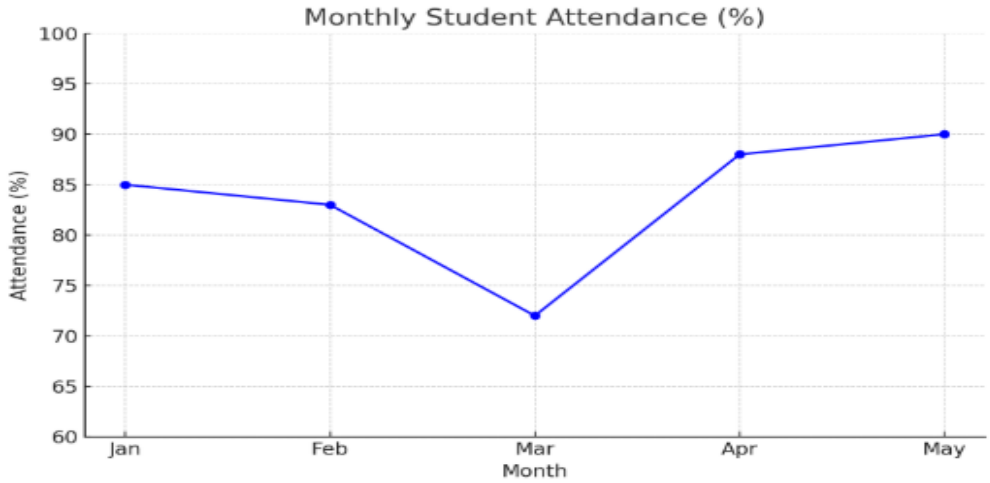
Sample Example: Student Attendance

Suppose Raw Data is:

1. Jan: 85%
2. Feb: 83%
3. Mar: 72%
4. Apr: 88%
5. May: 90%

Chart:

Line graph showing attendance dropping in March and rising again.



Story:

"March saw the lowest attendance due to board exam stress and a flu outbreak. The school responded with counseling and health sessions, leading to a strong recovery by May."

Output: Insight of data is delivered. The action taken by the school is justified.

Importance of Data Storytelling

1. Makes complex data easy to understand
2. Helps people remember insights
3. Encourages empathy and engagement
4. Drives better decisions

Steps to Create a Good Data Story

1. Define your purpose – What do you want to explain or prove?
2. Know your audience – What context do they need?
3. Choose the right data – Relevant, accurate, and timely.
4. Find the insight – What pattern or surprise stands out?
5. Use visuals smartly – Highlight the key points.
6. Build a narrative – Start with "why," explain "what," and end with "so what."

Real-World Use Cases

a) Business: "Customer churn increased after the new pricing policy."
b) Education: "Online learning improved grades in math but not science."
c) Healthcare: "Vaccination rates rose where outreach programs were active."
d) Policy: "Air quality improved 40% in cities with car-free zones."

Common Mistakes to Avoid

a) Overloading with charts
b) Ignoring the audience's level of understanding
c) Telling stories without valid data
d) Using visuals that confuse rather than clarify

1.4 Introduction to Inferential Statistics

Inferential statistics helps us make predictions or generalizations about a large group by studying a small group called a sample.

How it works:

Step 1: Collect a sample of data such as 100 students.

Step 2: Analyze it.

Step 3: Infer or guess what the result would be for the whole population such as, all 1000 students.

Key Concepts:

1. *Population:* The entire group such as, all students in school.
2. *Sample:* A small part of the population used to study the group.
3. *Hypothesis Testing:* A method to test if our guess or assumption is correct.
4. *Confidence Level:* Tells how sure we are about the result such as, 95% sure.

Example:

You survey 50 students and find that 80% prefer online notes. With inferential stats, you estimate that around 80% of all students may feel the same.

Here's a simple and practical example to explain Inferential Statistics.

Average Marks of School Students

A school has 1,000 students. You want to know the average math marks of all students, but checking every student's marks is time-consuming.

So, you randomly select a sample of 50 students and calculate:

a) Sample Mean ($\bar{x}$) = 72 marks

b) Standard Deviation (s) = 8 marks

Inferential statistics helps you use this sample data to make conclusions or predictions about the entire student population.

Example of Inference:

You use a statistical formula (like a confidence interval) and say:

"We are 95% confident that the true average math marks of all 1,000 students lies between 70 and 74 marks."

This is an inference — you're using data from 50 students to say something about all 1,000 students.

Common Inferential Tools:

Method	**Purpose**
Confidence Intervals	Estimate population parameters
Hypothesis Testing	Test assumptions (e.g., A/B testing)
t-tests/z-tests	Compare means
ANOVA	Compare multiple groups
Regression Analysis	Predict outcomes

Real-Life Uses of Inferential Statistics:

1. *Polling:* Predicting election results from a small voter sample.
2. *Medical Trials:* Estimating if a new drug works based on 200 patients.
3. *Quality Control:* Inferring product quality by testing 10 items out of 1,000.
4. *Education:* Predicting school wide performance using one classroom's data.

Practical Activities

Activity 4.1. Class Canteen Feedback Analysis (Descriptive Statistics)

Task: Collect ratings (1–10) from 30 students about canteen food taste.

Steps:

Step 1. Enter ratings in a table.

Step 2. Compute Mean, Median, Mode, and Range of ratings.

Step 3. Interpret:

- Mean → Overall satisfaction.
- Median → Typical feedback.
- Mode → Most common opinion.
- Range → Variation in opinions.

Outcome: Students learn how central tendency & spread summarize data.

Activity 4.2. Test Scores Summary

Data: Marks of 10 students in a test.

Example: 65, 70, 75, 70, 90, 80, 85, 60, 70, 95

Steps

Step 1. Compute Mean, Median, Mode, Range.

Step 2. Calculate Standard Deviation using the formula.

Discuss: Which measure is most useful for a teacher to understand performance?
Outcome: Students see how average vs. variability give different insights.

Activity 4.3. Visualization Lab

Task: Represent the same dataset in different graphs.

Dataset: Monthly sales (Jan–Jun) = [40, 55, 60, 50, 65, 70]

Steps

Step 1. Plot Bar Graph, Line Graph, Pie Chart, Histogram.

Step 2. Discuss which chart best shows comparison, trend, contribution, distribution.

Step 3. Show one misleading version (e.g., bar graph with Y-axis starting at 50).

Outcome: Students understand good vs. misleading visualizations.

Activity 4.4. Data Storytelling Roleplay

Task: Use attendance data: Jan=85%, Feb=83%, Mar=72%, Apr=88%, May=90.

Steps

Step 1. Plot a line graph.

Step 2. Groups create a short story explaining the trend (e.g., "March drop due to flu outbreak").

Step 3. Present stories to class.

Outcome: Students learn data → insight → narrative flow.

Activity 5: Inferential Statistics Simulation

Scenario: School has 200 students. Checking everything is time-consuming.

Steps

Step 1. Randomly select a sample of 20 students' marks.

Step 2. Compute sample mean & standard deviation.

Step 3. Use formula to build a 95% confidence interval for population mean.

Step 4. Compare with the actual average (teacher provides).

Outcome: Students see how sample → population inference works.

Activity 4.6. Hypothesis Testing Game

Scenario: Class A says "canteen food tastes better than Class B."

Steps

Step 1. Collect ratings (1–10) from 10 students each in Class A & B.

Step 2. Compute mean rating for each class.

Step 3. Perform a simple t-test (conceptual, without heavy math) to check if the difference is significant.

Step 4. Discuss possible conclusion: "Yes, Class A rated higher, but the difference may be due to chance."

Outcome: Students understand why inferential statistics is needed beyond averages.

Activity 4.7. Real-Life Connection Debate

Prompt: "Which is more useful in real life — Descriptive or Inferential Statistics?"

Group 1 argues with examples (e.g., summarizing school test scores).

Group 2 argues with examples (e.g., election surveys, medical trials).

Outcome: Students grasp applications of both approaches.

Summary

- Introduces descriptive statistics: mean, median, mode, range, standard deviation.
- Covers data visualization principles: clarity, accuracy, simplicity, proper labeling, correct chart choice (bar, pie, line, histogram).
- Warns about misleading visualizations (truncated axes, wrong scales, 3D effects, cherry-picking).
- Explains data storytelling (data + visuals + narrative) to communicate insights.
- Introduces inferential statistics: using samples to make predictions about populations (confidence intervals, hypothesis testing, t-tests, regression).
- Shows real-world applications: polling, medical trials, education, quality control.

Check Your Progress

A. Multiple Choice Questions

1. The average of all values is called (a) Mode (b) Median (c) Mean (d) Range
2. Which chart best shows part-to-whole relationships? (a) Pie chart (b) Line graph (c) Histogram (d) Bar graph
3. Which statistic shows how spread out the data is? (a) Mode (b) Standard deviation (c) Median (d) Mean
4. Inferential statistics involves (a) Summarizing data (b) Predicting from a sample (c) Data cleaning (d) Collecting surveys
5. A misleading chart can result from (a) Starting Y-axis at zero (b) Choosing proper chart types (c) Cherry-picking data (d) Adding clear labels

B. Fill in the Blanks

1. _____ is the value that appears most often in a dataset.
2. Range = _____ – Minimum.
3. Graphs and charts help in _____ of data.

4. Data storytelling combines data, visuals, and _____.
5. In inferential statistics, a small group studied is called a _____.

C. True or False

1. The median is the same as the mean in all datasets.
2. Line graphs are best for trends over time.
3. Misleading visuals can change how data is interpreted.
4. Inferential statistics uses all population data directly.
5. Standard deviation measures data spread around the mean.

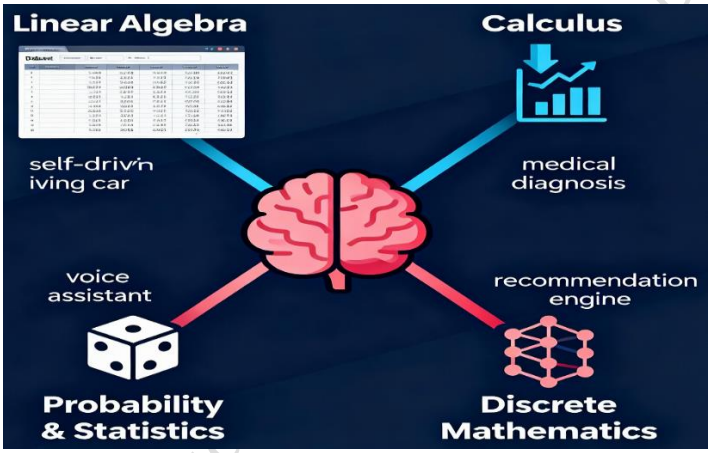
D. Short Answer Questions

1. Define mean, median, and mode.
2. What is the purpose of standard deviation?
3. List two principles of good data visualization.
4. What is data storytelling?
5. Give one real-world example of inferential statistics.

Module 4. Mathematics for AI

Module Overview

Artificial Intelligence may look magical on the surface, but behind it lies mathematics that makes machines think and learn. Linear algebra helps represent images, videos, and large datasets as vectors and matrices. Calculus makes AI "learn" by minimizing errors through optimization, much like adjusting your study strategy after checking your test scores. Probability and statistics allow AI to handle uncertainty—for example, predicting the weather or recommending movies based on your past choices. Discrete mathematics gives AI the logic and structure to reason, just like Google Maps uses graphs to find the shortest route. Together, these branches of mathematics form the foundation of AI, enabling real-world applications such as self-driving cars, medical diagnosis systems, recommendation engines, and voice assistants like Alexa or Siri.



Learning Outcomes

After completing this module, you will be able to:

- Understand key Linear Algebra concepts such as vectors, matrices, matrix multiplication, and their use in AI models.
- Explain the role of Calculus in optimization, including derivatives and gradient-based methods used to minimize errors in AI systems.
- Apply Probability and Statistics concepts such as probability rules, mean, variance, distributions, and hypothesis testing in AI applications.
- Understand Discrete Mathematics concepts like logic, sets, relations, functions, and graph theory used in AI algorithms.
- Recognize the importance of mathematical foundations in developing and analyzing AI models.

Module Structure

Session 1. Linear Algebra Concepts Used in AI

Session 2. Calculus for Optimization

Session 3. Probability and Statistics in AI

Session 4. Discrete Mathematics for AI

Session 1. Linear Algebra Concepts Used in AI

Linear algebra provides the language of AI. Concepts such as vectors, matrices, and tensors are used to represent data, perform transformations, and model relationships. Operations like matrix multiplication, transpose, and eigenvalues are crucial for handling datasets, images, and neural network computations.

Imagine you're designing a photo filter app like Instagram. Behind the scenes, your app uses Linear Algebra to apply filters, rotate images, compress data, and more. A photo filter app like Instagram, Snap seed, or VSCO uses mathematical operations and machine learning to change how images look. Figure 1 shows linear algebra in photo filters. In this session we will discuss linear algebra concepts used in AI.

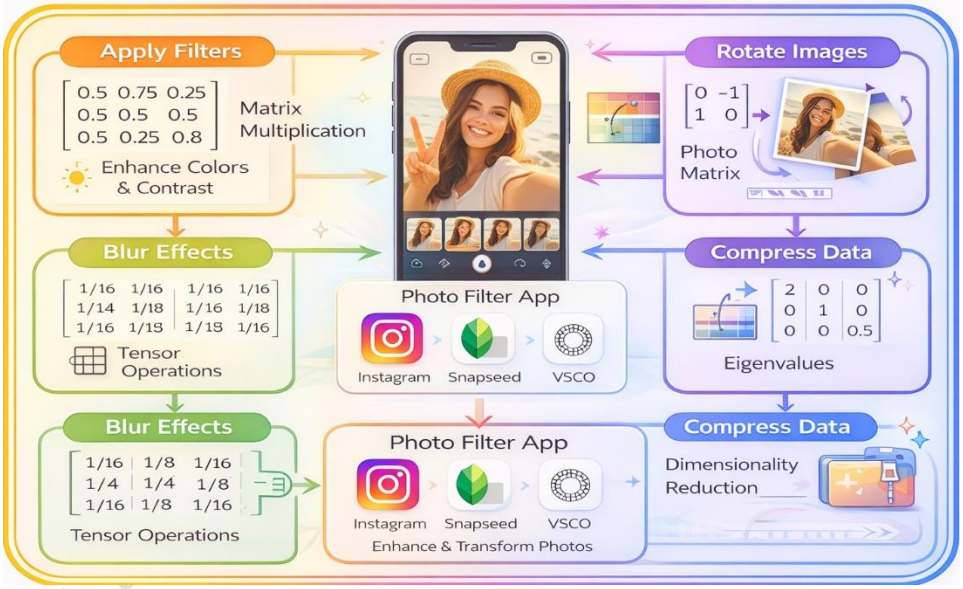


Fig. 1.1: Linear Algebra in Photo Filters

1.1 Linear Algebra Basics

Linear algebra is the mathematics of vectors, matrices, and linear transformations, and it is the foundation of many AI, machine learning, and image processing applications.

Vectors: A vector is an ordered list of numbers. It represents:

1. A point in space
2. A direction and magnitude
3. A set of features in machine learning

Example: A 3D vector is shown below.

$$\vec{v} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$$

The dimension of a vector is the number of components in the vector. Operations such as addition, scalar multiplication and dot product can be performed on vectors.

Real Use of Vectors:

1. In AI feature vectors represent input data.
2. In physics, the direction of force is a vector.
3. In images pixel color (R, G, B) is represented by a vector.

Matrices: A matrix is a rectangular array of numbers with rows and columns.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Here the matrix size is 2 rows × 3 columns (2×3). They are used to represent linear transformations, systems of equations and images

Real Use of Matrices:

1. Gray scale images are matrices of brightness values as shown in Figure 2.
2. Neural network weights are matrices
3. Databases can be represented by a matrix of rows and columns, where rows are records and columns are features.

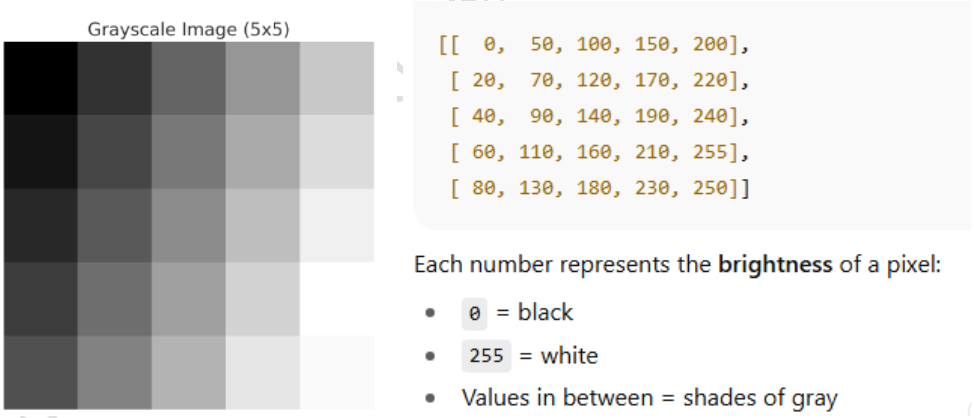


Fig. 1.2: Gray Image and its Matrix

Tensors: A tensor is a multi-dimensional array. In tensor 0D tensor is a scalar, 1D tensor is a vector, 2D tensor is a matrix and 3D+ tensors is a data structure for images and videos.

Example:

A color image (RGB) is a tensor of shape: Height × Width × 3

Real Use of Tensors:

1. Images and video in AI models are represented by tensors as shown in Figure 1.3.
2. Deep learning frameworks like TensorFlow and PyTorch process tensors.

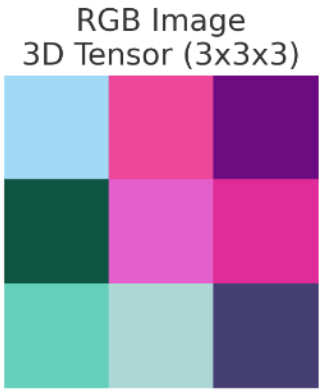


Fig. 1.3: RGB Image

1.2 Matrix Multiplication

This operation is not element-wise. You multiply rows of the first matrix by columns of the second.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \Rightarrow AB = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Rules of matrix multiplication:

1. A must have as many columns as B has rows.
2. Result is the new matrix

Real Use of Matrix Multiplication:

1. Matrix multiplication is used in feed-forward operations in neural networks.
2. Matrix multiplication is used in applying filters to images.

Matrix Transposition

Transpose of a matrix flips rows and columns (Rows ↔ Columns).

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Real Use of matrix transposition:

1. Reformatting data is achieved through matrix transposition.
2. Changing orientation of data is achieved by matrix transposition.
3. Matrix transposition is required in dot product and other calculations.

1.2 Eigenvalues and Eigenvectors

An eigenvector is a direction that remains unchanged during a linear transformation, and the eigenvalue is how much it is stretched or compressed. Imagine pushing on a rubber sheet from all sides (linear transformation). Some lines on the sheet don't rotate, they just stretch or compress. Those directions are eigenvectors and how much they stretch are eigenvalues. It is represented as:

$$Av = \lambda v$$

Where:

A = matrix

v = eigenvector

λ = eigenvalue

Real Use of Eigenvalues:

1. It is used in Principal Component Analysis (PCA) for dimensionality reduction
2. Face recognition makes use of eigenfaces.
3. Stability analysis in systems makes use of eigenvalues.

The word "Eigen" comes from German, where it means "own", "proper", or "characteristic". An eigenvector is a vector that represents a “proper direction” of a transformation, that is, it does not change direction, only scales. An eigenvalue is the “characteristic amount” by which the eigenvector is stretched or shrunk.

Simple Example

Let's use this matrix:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

Try this vector:

$$\vec{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Apply matrix:

$$A\vec{v} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} = 2\vec{v}$$

So $\vec{v}$ is an **eigenvector**, and **2** is the **eigenvalue**.

1.3 Linear Transformations

A linear transformation changes a vector space while preserving vector addition and scalar multiplication.

It is represented by:

$$T(\vec{x}) = A\vec{x}$$

Where A is the transformation matrix.

Examples of linear transformation:

1. Rotation of a vector
2. Scaling of a vector
3. Shearing of a vector

Rotation, Scaling, and Shearing are three important types of linear transformations in linear algebra. These transformations change the shape, size, or orientation of vectors or images, and are widely used in computer graphics, robotics, AI, and image processing.

Rotation

Rotation turns a shape or a vector around a point, usually the origin without changing its length. The direction remains the same, but its orientation changes.

In 2D space, to rotate a point or vector by a certain angle (like 90 degrees), we use a rotation matrix. When this matrix is applied, the object spins but doesn't stretch or shrink.

Example:

Imagine a point (1, 0). Rotating it 90 degrees counter clockwise moves it to (0, 1). Its length stays the same, only its direction changes.

Matrix for 2D Rotation (by angle θ):

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Example:

Rotate vector $\vec{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ by 90°:

$$R(90^\circ) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Real-life Uses:

1. Rotating shapes in games and animation
2. Turning a robot or drone to face a new direction
3. Rotating images in photo editing software

Scaling

Scaling changes the size of an object or vector. It either enlarges or reduces the shape, depending on the scale factor used. The object stays in the same direction, but its dimensions change.

You multiply each coordinate by a scale factor. For example:

Scaling by 2 makes the object twice as big.

Scaling by 0.5 makes it half the size.

Matrix for Scaling:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

- s_x: scale factor for x-axis
- s_y: scale factor for y-axis

Example:

Scale vector $\vec{v} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ by 2 in x, 0.5 in y:

$$S = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix} \Rightarrow S\vec{v} = \begin{bmatrix} 4 \\ 1.5 \end{bmatrix}$$

Scaling can be done equally in all directions (uniform scaling) or differently along x and y axes (non-uniform scaling).

Example:

A point (2, 3) scaled by 2 along x and 0.5 along y becomes (4, 1.5). The shape is now wider and shorter.

Real-life Uses:

1. Zooming in and out of images
2. Resizing design elements in software
3. Enlarging small icons on a mobile screen

Shearing

Shearing is a transformation that slants the shape of an object. It shifts one part of the object more than another, like pushing the top of a rectangle sideways while keeping the bottom fixed.

In horizontal shearing, the x-coordinate of each point is changed in proportion to its y-coordinate. In vertical shearing, the y-coordinate is changed in proportion to the x-coordinate.

Example:

Start with a point at (2, 3). Apply a horizontal shear that adds the y-value to the x-value. The new point becomes (5, 3). The object looks tilted or slanted.

$$\text{Shear}_x = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$$

Shear matrix in Y-direction:

$$\text{Shear}_y = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix}$$

- k: shear factor

Example:

Apply X-shear with $k = 1$ to vector $\vec{v} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$:

$$\text{Shear}_x \vec{v} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2+3 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

Real-life Uses:

1. Creating italic text or slanted letters
2. Simulating 3D motion or blur effects in images
3. Applying perspective in computer graphics

Real-World Applications of Linear Algebra

Concept	Real-World Use
Vectors	Representing data, directions
Matrices	Images, equations, neural networks
Tensors	Deep learning models, image/video input
Matrix Multiplication	Neural network layers, filters
Transposition	Data preparation, mathematical operations
Eigenvalues/vectors	PCA, face recognition, compression

Concept	Real-World Use
Linear Transformation	Graphics, camera effects, simulations

Practical Activities

Activity 1.1. Vector Features in AI

Give students a simple dataset:

Student 1: [90 (Maths), 80 (Science), 85 (English)]

Student 2: [70, 95, 60]

Student 3: [88, 76, 92]

Ask them to treat each student's marks as a vector.

Tasks:

1. Compute the magnitude of each student's vector.
2. Find the dot product between Student 1 and Student 3 (similarity).
3. Discuss how this relates to comparing feature vectors in AI (e.g., recommendation systems).

Answers:

a) $|v_1| = \sqrt{(90^2 + 80^2 + 85^2)} = \sqrt{(8100 + 6400 + 7225)} = \sqrt{21725} \approx 147.4$

b) $v_1 \cdot v_3 = (90\times88) + (80\times76) + (85\times92) = 7920 + 6080 + 7820 = 21{,}820$

This shows how vectors measure similarity of features like comparing students or user preferences in AI systems.

Activity 1.2. Image as a Matrix

Show a 5×5 grayscale image matrix with pixel values (0–255). Example:

[[100, 120, 130, 140, 150],
[90, 110, 120, 130, 140],
[80, 100, 110, 120, 130],
[70, 90, 100, 110, 120],
[60, 80, 90, 100, 110]]

Tasks:

- Transpose the matrix → flip rows and columns (students see how the image rotates).
- Multiply the matrix by 2 (clipping above 255) → image looks brighter.
- Multiply by 0.5 → the image looks darker.

Connect to: Instagram brightness filters.

Answers:

a) Transpose flips rows and columns:

[[100, 90, 80, 70, 60],
[120,110,100, 90, 80],
[130,120,110,100, 90],
[140,130,120,110,100],

[150,140,130,120,110]]

b) Multiply ×2 (clip >255):

[[200,240,255,255,255],
[180,220,240,255,255],
[160,200,220,240,255],
[140,180,200,220,240],
[120,160,180,200,220]]

c) Multiply ×0.5:

[[50, 60, 65, 70, 75],
[45, 55, 60, 65, 70],
[40, 50, 55, 60, 65],
[35, 45, 50, 55, 60],
[30, 40, 45, 50, 55]]

This is how brightness/darkness filters in photo apps work.

Activity 1.3. Matrix Multiplication as Filter

Provide a 3×3 image patch and a filter kernel (e.g., edge detection):

Image = [[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]

Filter = [[-1, -1, -1],
[-1, 8, -1],
[-1, -1, -1]]

Ask students to compute the matrix multiplication (convolution) result.

Connect to: how AI applies filters in computer vision.

Answer:

(1×-1 + 2×-1 + 3×-1 + 4×-1 + 5×8 + 6×-1 + 7×-1 + 8×-1 + 9×-1)

= (-1 -2 -3 -4 + 40 -6 -7 -8 -9) = 0

Result shows filter highlights edges; AI uses this in CNNs for vision.

Activity 1.4. Eigenfaces for Face Recognition

Show students a set of simple faces.

Explain that AI doesn't store the whole face but key features (eigenfaces).

Tasks:

- Give them 3×3 matrices (simplified faces).
- Ask them to compute variance along rows/columns.
- Discuss how eigenvectors capture the "most important features" (e.g., eyes,

mouth).

Answer:

Given faces as 3×3 matrices:

Face A = [[1,0,1],
[0,1,0],
[1,0,1]]

Face B = [[1,1,1],
[0,1,0],
[1,1,1]]

Face A row sums: [2,1,2]

Face B row sums: [3,1,3]

The "main features" are in rows 1 & 3 (eyes, mouth).

Eigenvectors capture the main "features" → eigenfaces in AI.

Activity 1.5. Linear Transformations on Shapes

Give students a set of 2D points (a square):

(0,0), (1,0), (1,1), (0,1)

Tasks:

- Apply rotation matrix (90°) → draw a new square.
- Apply scaling (×2) → bigger squares.
- Apply shearing → slanted square.

Connect to:

1. Rotating photos in editing apps
2. Zoom in/out in image viewers
3. Italic text or perspective effects

Answers:

a) Rotation matrix [[0,-1],[1,0]]:

(0,0) → (0,0), (1,0) → (0,1), (1,1) → (-1,1), (0,1) → (-1,0)

b) Scaling by 2:

(0,0),(2,0),(2,2),(0,2)

c) Shear: (x,y) → (x+y,y)

(0,0)→(0,0), (1,0)→(1,0), (1,1)→(2,1), (0,1)→(1,1)

Shows image rotation, zooming, italics in apps.

Activity 1.6. Tensor in Images and Videos

Tasks:

a) Grayscale 28×28 image → shape?

b) RGB 28×28 image → shape?

c) 10-frame RGB video (28×28) → shape?

Answers:

a) (28, 28) = 2D tensor (matrix)

b) (28, 28, 3) = 3D tensor
c) (10, 28, 28, 3) = 4D tensor
AI models in TensorFlow/PyTorch process such tensors.

Python Programs Activity

Activity 1: Vector Operations (Addition, Scalar Multiplication, Dot Product)

Use in AI: Feature vectors in machine learning, similarity (dot product) in recommendation systems, and image color channels.

Python Program:

```
import numpy as np

# Define two vectors
v1 = np.array([2, 3, 4])
v2 = np.array([1, 0, -1])

# Vector addition
v_add = v1 + v2

# Scalar multiplication
v_scalar = 3 * v1

# Dot product
v_dot = np.dot(v1, v2)

print("Vector 1:", v1)
print("Vector 2:", v2)
print("Vector Addition:", v_add)
print("Scalar Multiplication (3*v1):", v_scalar)
print("Dot Product:", v_dot)
```

Sample Output:

```
Vector 1: [2 3 4]
Vector 2: [ 1  0 -1]
Vector Addition: [3 3 3]
Scalar Multiplication (3*v1): [ 6  9 12]
Dot Product: -2
```

Activity 1.2. Matrix Operations (Multiplication & Transpose)

Use in AI: Matrix multiplication = forward pass in neural networks. Transpose = data reshaping and dot product calculations.

Python Program:

```
# Define two matrices
A = np.array([[1, 2, 3],
              [4, 5, 6]])
```

```
B = np.array([[1, 4],
              [2, 5],
              [3, 6]])

# Matrix multiplication
C = np.dot(A, B)

# Transpose of A
A_T = A.T

print("Matrix A:\n", A)
print("Matrix B:\n", B)
print("Matrix Multiplication (A*B):\n", C)
print("Transpose of A:\n", A_T)
```

Sample Output:

```
Matrix A:
 [[1 2 3]
 [4 5 6]]
Matrix B:
 [[1 4]
 [2 5]
 [3 6]]
Matrix Multiplication (A*B):
 [[14 32]
 [32 77]]
Transpose of A:
 [[1 4]
 [2 5]
 [3 6]]
```


Activity 1.3. Linear Transformation (Rotation and Scaling of a Vector)

Use in AI:

1. Rotations & scalings are used in image transformations, robotics, and computer graphics.
2. Essential for data augmentation in deep learning.

Python Program:

```
# Define a vector (point)
v = np.array([1, 0])

# Rotation matrix (90 degrees counterclockwise)
theta = np.pi / 2
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
```

```
                            [np.sin(theta),  np.cos(theta)]])

# Scaling matrix (scale x by 2, y by 0.5)
scaling_matrix = np.array([[2, 0],
                           [0, 0.5]])

# Apply transformations
v_rotated = np.dot(rotation_matrix, v)
v_scaled = np.dot(scaling_matrix, v)

print("Original Vector:", v)
print("Rotated Vector (90°):", v_rotated)
print("Scaled Vector (x2, y0.5):", v_scaled)
```

Sample Output:

```
Original Vector: [1 0]
Rotated Vector (90°): [0. 1.]
Scaled Vector (x2, y0.5): [2. 0.]
```


Summary

Linear Algebra is the foundation of Artificial Intelligence. It provides mathematical tools such as vectors, matrices, and tensors to represent and process data.

Operations like matrix multiplication, transpose, and eigenvalues help in transforming images, handling datasets, and performing neural network computations.

In applications like photo filter apps, linear algebra is used to rotate images, apply filters, enhance colors, and compress data. These concepts enable AI systems to analyze patterns and make intelligent decisions efficiently.

Check Your Progress

A. Multiple Choice Questions

1. Which of the following is a scalar quantity? (a) [2, 3] (b) 5 (c) (1, 2, 3) (d) [[1, 0], [0, 1]]
2. The dot product of vectors [1, 2] and [3, 4] is (a) 7 (b) 10 (c) 11 (d) 14
3. Which of the following is an identity matrix of order 2? (a) [[1, 1], [1, 1]] (b) [[1, 0], [0, 1]] (c) [[0, 1], [1, 0]] (d) [[2, 0], [0, 2]]
4. The transpose of [[1, 2], [3, 4]] is (a) [[1, 3], [2, 4]] (b) [[1, 2], [4, 3]] (c) [[2, 1], [3, 4]] (d) [[4, 3], [2, 1]]
5. In AI, matrices are mainly used for (a) Web browsing (b) Storing and transforming data (c) Music recording (d) Random guessing

B. Fill in the Blanks

1. A _____ is an ordered list of numbers that represent magnitude and direction.

2. The product of a matrix and its transpose is always a _____ matrix.
3. The _____ product of vectors is also known as the inner product.
4. A matrix with equal number of rows and columns is called a _____ matrix.
5. In AI, image data is often represented in the form of a _____.

C. True or False

1. Vectors can only have two dimensions.
2. Identity matrix multiplied with any matrix gives the same matrix.
3. The transpose of a matrix is obtained by flipping it along the diagonal.
4. Scalar multiplication means multiplying two matrices.
5. Linear algebra is essential in neural networks and machine learning.

D. Short Answer

1. Define a vector with an example.
2. What is the difference between a row vector and a column vector?
3. Explain the significance of an identity matrix in AI.
4. What does the dot product of two vectors represent?
5. Why is linear algebra important in AI applications?

Session 2. Calculus for Optimization

Calculus powers the learning process of AI. Derivatives, gradients, and optimization techniques like Gradient Descent help in minimizing error functions and improving model accuracy. Understanding differentiation, integration, and back-propagation enables the training of complex AI models.

You can optimize your daily study time with calculus. Imagine you're preparing for final exams and want to maximize your test score by choosing the optimal number of hours to study each day. But there's a catch. If you study too little, you won't be prepared. If you study too much, you get tired, and your performance drops. So, there must be an optimal study time where your performance (score) is highest. In step 1 you can model the problem with a function. In step 2 you can use derivatives to find the maximum. Finally, you observe that you should study 5 hours per day to maximize your score. Any more or less, and your performance will decline — as predicted by calculus. Figure 1 shows balancing study time.

This is Calculus for Optimization — finding the best possible value, maximum or minimum of a function using derivatives. At the core of AI is learning from errors. Machines try to improve performance by minimizing mistakes — and calculus tells us how to change the internal parameters like weights in neural networks to do that. In this session we will study this calculus.

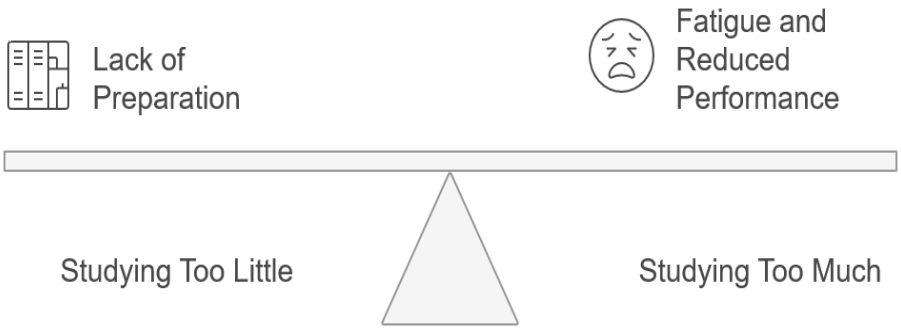


Fig. 2.1: Balancing Study Time

2.1 Limits and Derivatives

Limit: A limit is the value a function approaches as the input gets closer to a particular point.

Example:

$$\lim_{x \to 2}(3x + 1) = 7$$

This means as x gets closer to 2, the function 3x + 1 gets closer to 7.

Limits are the foundation of derivatives in calculus — they help measure change.

Derivative: A derivative represents the rate of change of a function with respect to a variable.

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

It tells us how much the output f(x) changes when we change input x slightly.

In real life derivatives are used for various purposes such as:

1. The derivative of distance is equal to speed.
2. The derivative of speed is equal to acceleration.
3. In AI models such as neural networks, the derivative tells us how output changes when we change weights or inputs

2.2 Gradient and Partial Derivatives

Partial Derivative: A partial derivative measures how a multi variable function changes when only one variable is changed, keeping all others constant. It is used when a function depends on more than one variable.

Example: Consider a function,

$$f(x, y) = 3x^2 + 2y$$

Partial derivatives:

$$\frac{\partial f}{\partial x} = 6x, \quad \frac{\partial f}{\partial y} = 2$$

These show how the function changes as each variable change individually, keeping the other constant.

Gradient: The gradient is a vector of all partial derivatives of a function.

For a function f(x,y,z), the gradient is:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right]$$

It points in the direction of the steepest increase of the function — used in training ML models.

Gradient Descent Optimization: Gradient Descent is an optimization algorithm used to minimize a function, especially a loss function in machine learning.

Steps of working:

d) Start with initial parameters like weights in a model.
e) Calculate the gradient, that is, slope.
f) Move in the opposite direction of the gradient to reduce error.
g) Repeat until error is minimized.

Update Rule:

$$\theta = \theta - \eta \cdot \nabla L(\theta)$$

Where:

θ = model parameters or weights

η = learning rate

∇L(θ) = gradient of the loss function

Think of it like a person climbing down a hill with eyes closed, taking steps based on slope direction.

Simple Analogy: Walking Down a Hill

Imagine you're blindfolded and trying to reach the lowest point in a valley (minimum error). You:

- Feel the slope under your feet (gradient)
- Take small steps downhill (update weights)
- Stop when it's flat (minimum)

This is how gradient descent works — and calculus makes it possible.

Chain Rule and Back-propagation

Chain Rule: The Chain Rule is a rule in calculus for finding the derivative of a composite function — that is, when one function is inside another. It is used to find the derivative of a composite function:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

Example: If,

$$y = f(g(x)) \Rightarrow \frac{dy}{dx} = f'(g(x)) \cdot g'(x)$$

This is crucial in neural networks where multiple functions are composed in layers.

Back-propagation: Back-propagation is an algorithm that uses the chain rule to compute the gradient of the loss function with respect to each parameter in a neural network.

Steps:

- Forward pass: Compute the output
- Compute loss
- Backward pass: Use the chain rule to compute gradients layer by layer
- Update weights using gradient descent

It's how AI learns from mistakes — adjusting weights to improve predictions.

Table

Concept	**Definition / Use**
Limit	Approach of a function as input nears a value
Derivative	Instantaneous rate of change
Partial Derivative	Change wrt one variable in multi-variable function
Gradient	Vector of partial derivatives
Gradient Descent	Method to minimize error/loss
Chain Rule	Rule to differentiate composite functions
Back-propagation	Uses chain rule to adjust weights in neural networks

Real-World Applications of Calculus

- Self-driving cars make use of back-propagation to train models to detect lanes, and objects.
- The face recognition system optimizes error using gradient descent.
- Stock predictions systems use derivatives to help understand changing trends.
- Health apps adjust predictions by learning from user data.

Figure 2.2 shows real world applications of calculus.

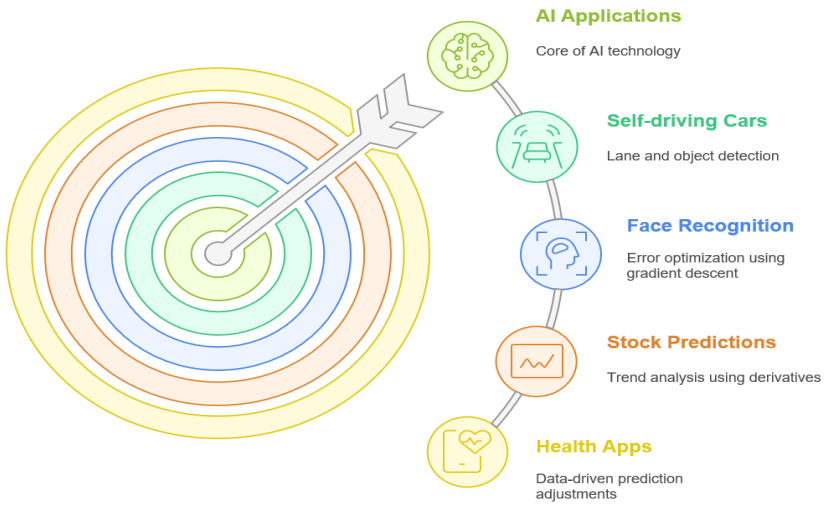


Fig. 2.2: Real World Applications of Calculus

Calculus is the mathematical engine behind AI learning. Without it, AI wouldn't be able to:

1. Understand change
2. Learn from data
3. Optimize decisions
4. Train neural networks

It transforms raw numbers into smart predictions.

Practical Activity

Activity 2.1. Compute Derivatives of Simple Cost Functions by Hand

Example: Cost Function (Mean Squared Error)

Let's take a simple cost (loss) function for one training example:

$$C(w) = (w\,x - y)^2$$

Where:

w is the weight parameter we are optimizing

x is the input feature

y is the actual output (label)

Compute the Derivative (Gradient) with Respect to w

$$C(w) = (w\,x - y)^2$$

Use chain rule:

Let:

$u = w\,x - y$

So, $C = u^2$

Then:

Then:

$$\frac{dC}{dw} = \frac{dC}{du} \cdot \frac{du}{dw}$$

Step-by-step:

1. $\frac{dC}{du} = 2u = 2(wx - y)$
2. $\frac{du}{dw} = x$

Therefore:

$$\frac{dC}{dw} = 2(wx - y) \cdot x$$

This is the gradient (slope) of the cost function with respect to weight `w`.

This is the gradient (slope) of the cost function with respect to weight w.

Activity 2.2: Implement Gradient Descent in Python

Problem: Minimize the Cost Function

We'll use:

$C(w) = (w\,x - y)^2$

Goal: Find the best w using gradient descent.

Python Code:

```
# Simple Gradient Descent for One Variable Linear Model
# Data point
x = 2
y = 5
# Initial weight
w = 0.0
# Learning rate
lr = 0.1
# Number of iterations
epochs = 20
for i in range(epochs):
    # Forward pass: compute prediction
    y_pred = w * x

    # Compute loss
    loss = (y_pred - y) ** 2

    # Compute gradient
    grad = 2 * (w * x - y) * x

    # Update weight using gradient descent
    w = w - lr * grad

    # Print result at each step
    print(f"Epoch {i+1}: w = {w:.4f}, loss = {loss:.4f}")
```

Output (example):

```
Epoch 1: w = 2.0000, loss = 25.0000
Epoch 2: w = 2.8000, loss = 1.0000
Epoch 3: w = 3.2800, loss = 0.0640
...
```

Activity 2.3. Derivative – Rate of Change

Use in AI: Measures how model output changes with respect to inputs or weights. Used in optimization and training of AI models.

Python Program:

```
import sympy as sp
# Define variable and function
x = sp.symbols('x')
f = 3*x**2 + 2*x + 1

# Derivative
f_prime = sp.diff(f, x)
```

```
print("Function:", f)
print("Derivative:", f_prime)

# Evaluate derivative at x = 2
slope_at_2 = f_prime.subs(x, 2)
print("Slope at x=2:", slope_at_2)
```

Sample Output:

```
Function: 3*x**2 + 2*x + 1
Derivative: 6*x + 2
Slope at x=2: 14
```

Activity 2.4. Optimization – Best Study Hours

Use in AI: Shows how to find maxima/minima of performance functions.
Similar to how AI finds the best model parameters.

Python Program:

```
# Function: performance = - (x-5)^2 + 25
x = sp.symbols('x')
performance = - (x-5)**2 + 25

# Derivative
derivative = sp.diff(performance, x)

# Solve derivative = 0 for maxima
optimal_hours = sp.solve(derivative, x)
print("Optimal Study Hours:", optimal_hours[0])
```

Sample Output:

```
Optimal Study Hours: 5
```

Activity 2.5. Partial Derivatives – Multi-variable Functions

Use in AI: Used in functions with many variables (weights).
Helps compute gradients in neural networks.

Python Program:

```
x, y = sp.symbols('x y')
f = x**2 + y**2 + 3*x*y
# Partial derivatives
df_dx = sp.diff(f, x)
df_dy = sp.diff(f, y)
print("f(x,y) =", f)
print("∂f/∂x =", df_dx)
print("∂f/∂y =", df_dy)
```

Sample Output:

```
f(x,y) = x**2 + y**2 + 3*x*y
∂f/∂x = 2*x + 3*y
∂f/∂y = 2*y + 3*x
```

Activity 2.6. Gradient Descent – Error Minimization

Use in AI: Core algorithm for training machine learning models.

Minimizes loss/error by updating weights step by step.

Python Program:

```
import numpy as np
# Function: f(x) = (x-3)^2
def f(x):
    return (x-3)**2
# Derivative: f'(x) = 2(x-3)
def df(x):
    return 2*(x-3)
# Gradient Descent
x = 10       # initial guess
lr = 0.1     # learning rate
for i in range(10):
    x = x - lr * df(x)
    print(f"Iteration {i+1}: x = {x:.4f}, f(x) = {f(x):.4f}")
```

Sample Output:

```
Iteration 1: x = 8.6000, f(x) = 31.3600
Iteration 2: x = 7.2800, f(x) = 18.1504
Iteration 3: x = 6.0240, f(x) = 9.1496
Iteration 4: x = 4.8192, f(x) = 3.3080
Iteration 5: x = 3.6554, f(x) = 0.4295
Iteration 6: x = 3.3243, f(x) = 0.1055
Iteration 7: x = 3.1297, f(x) = 0.0168
Iteration 8: x = 3.0518, f(x) = 0.0027
Iteration 9: x = 3.0207, f(x) = 0.0004
Iteration 10: x = 3.0083, f(x) = 0.0001
Approaches x ≈ 3, the minimum point.
```


Activity 2.7. Chain Rule – Composite Function

Use in AI: Backbone of backpropagation in neural networks.

Allows AI models to update weights across multiple layers.

Python Program:

```
x = sp.symbols('x')
f = sp.sin(x**2)
```

```
# Derivative using chain rule
f_prime = sp.diff(f, x)
print("f(x) = sin(x^2)")
print("f'(x) =", f_prime)
```

Sample Output:

```
f(x) = sin(x^2)
f'(x) = 2*x*cos(x**2)
```

Demonstrates chain rule, crucial for back-propagation in neural networks.

Summary

Calculus plays a vital role in Artificial Intelligence, especially in optimization. It helps AI models learn by minimizing errors and improving performance.

Concepts like derivatives and gradients are used to measure how changes in input affect the output. In machine learning, algorithms use Gradient Descent to find the minimum error (loss) by adjusting model parameters step by step.

For example, when training a neural network, calculus helps update weights to make predictions more accurate. Thus, calculus enables AI systems to learn efficiently and continuously improve.

Check Your Progress

A. Multiple Choice Questions

1. The derivative of f(x)=x2 is (a) 2x (b) x (c) x3 (d) 2
2. In AI, derivatives are mainly used for (a) Data storage (b) Optimization and learning (c) Image compression (d) Encryption
3. Gradient Descent is used to (a) Maximize error (b) Minimize cost function (c) Store matrices (d) Increase learning rate
4. The slope of a tangent line at a point is given by (a) Integral (b) Gradient (c) Derivative (d) Transpose
5. The integral of f(x)=2x is (a) x2+C (b) 2x2 (c) x+C (d) 2+C

B. Fill in the Blanks

1. The process of finding the derivative is called _____.
2. The process of finding the area under a curve is called _____.
3. Gradient Descent updates weights in the _____ direction of the gradient.
4. A point where the derivative is zero may be a minimum, maximum, or a _____ point.
5. The derivative of a constant is always _____.

C. True or False

1. Gradient Descent is an optimization algorithm.

2. The derivative of x3 is 3x2.
3. Integration is the inverse process of differentiation.
4. Derivatives are not used in neural networks.
5. Optimization in AI only requires integration.

Session 3. Probability and Statistics in AI

AI often deals with uncertainty, which is managed using probability and statistics. Concepts such as conditional probability, Bayes' theorem, distributions, variance, and expected value are applied in prediction, decision-making, and probabilistic modeling. These tools allow AI systems to reason and adapt in uncertain environments.

You receive hundreds of emails daily, and your email app automatically detects and moves spam emails to the spam folder. This is achieved by using AI. It uses probability and statistics to decide whether this email is spam based on its words and features. Several other AI applications such as face recognition, word prediction, risk of a disease and recommendation system also make use of probability. Figure 3.1 shows email filtering using AI. In this session we will discuss probability and other statistical tools used in AI.

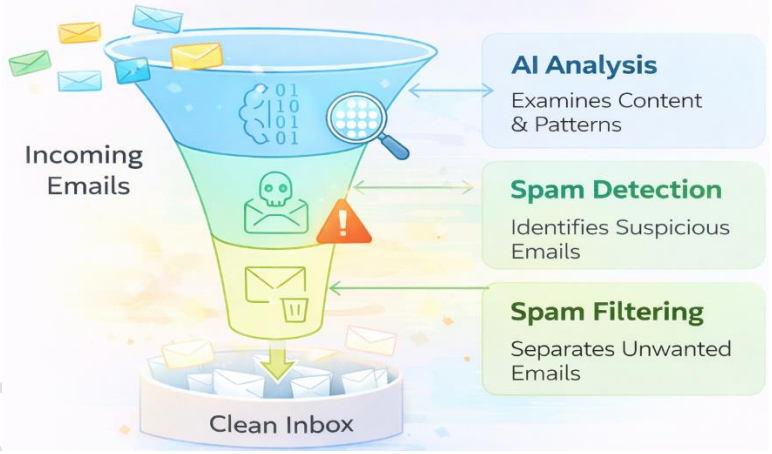


Fig. 3.1: Email filtering using AI

3.1 Basic Probability Theory

Probability is the mathematical language of uncertainty. It helps us make predictions and decisions when outcomes are not guaranteed, which is the case in most AI applications.

Probability

Probability is a number between 0 and 1 that represents how likely an event is to happen.

If P(A) = 1 → event A definitely happens

If P(A) = 0 → event A definitely does not happen

If P(A) = 0.5 → A is equally likely to happen or not

Types of Probability: There are three main types of probability as given below:

1. Theoretical Probability
2. Empirical (Experimental) Probability
3. Subjective Probability

1. Theoretical Probability

Probability based on logical reasoning and mathematical formulas, assuming all outcomes are equally likely.

Formula:

$$P(A) = \frac{\text{Number of favorable outcomes}}{\text{Total number of possible outcomes}}$$

Example:

Tossing a fair die:

1. Total outcomes = 6 (1 to 6)
2. Probability of getting 4 is P (getting a 4) = 1/6

Use in AI:

It is rarely used directly, but forms the basis of probability theory, which underlies Bayes' Theorem and Markov Chains.

2. Empirical (Experimental) Probability

Probability based on data or experimentation. It's calculated from the actual frequency of events occurring in past observations.

Formula:

$$P(A) = \frac{\text{Number of times A occurred}}{\text{Total number of trials}}$$

Example:

Out of 1,000 emails:

1. 200 contain the word "offer"
2. 150 of them were spam

P(email is spam | "offer") = 150/200} = 0.75200150 = 0.75

Use in AI:

This probability is used extensively in:

a) Spam detection
b) Sentiment analysis
c) Recommender systems
d) Predictive modeling

AI systems learn from past data and update probabilities as more data arrives, there empirical probability is in action.

3. Subjective Probability

Probability based on personal judgment, intuition, or expert opinion, not on actual data or mathematical calculation.

Example:

A doctor may say:

"There's an 80% chance this patient has malaria". It is based on symptoms and experience and not lab data.

Use in AI: This probability is used in many applications as given below:

1. Used in expert systems
2. Fuzzy logic systems
3. AI models where complete data is not available or only human estimates exist

Comparison Table

Type	Basis	Example	Role in AI
Theoretical	Known outcomes & logic	Dice, coins	Forms mathematical foundations
Empirical	Observed data	75% of emails with "free" are spam	Used in training ML models
Subjective	Beliefs or intuition	"80% chance of recession"	Used in decision support systems

3.2. Conditional Probability

Definition: The probability of event A happening given that event B has already occurred.

Written as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Real-World Example:

Let:

A = person has flu

B = person has fever

$$P(\text{Flu}|\text{Fever}) = \frac{P(\text{Flu and Fever})}{P(\text{Fever})}$$

In AI, this is often used in:

1. Medical diagnosis AI
2. Spam filters
3. Voice recognition

3.3 Bayes' Theorem

Bayes' Theorem reverses conditional probability. It allows us to find the probability of a cause, given an effect.

Formula:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where:

1. P(A|B): Probability of hypothesis A given evidence B
2. P(B|A): Probability of seeing evidence B if A is true
3. P(A): Prior probability of A
4. P(B): Total probability of B

Example: Email Spam Detection

Let:

A = email is spam

B = email contains word "win"

From historical data:

- $P(\text{Spam}) = 0.4$
- $P(\text{"win"}|\text{Spam}) = 0.8$
- $P(\text{"win"}) = 0.5$

Using Bayes' Theorem:

$$P(\text{Spam}|\text{"win"}) = \frac{0.8 \times 0.4}{0.5} = 0.64$$

→ There's a **64% chance** the email is spam if it contains "win".

There's a 64% chance the email is spam if it contains "win".

Applications in AI and ML

Application	**How Probability Helps**
Spam Filtering	Bayes classifiers use word probabilities
Speech Recognition	Predict next sound/word using probability
Medical Diagnosis	Probability of disease given symptoms
Recommendation Systems	Predict user choices using conditional probability
Weather Forecasting	Based on conditional trends from past data

3.4 Random Variable

A Random Variable (RV) is a numerical outcome of a random experiment. A random experiment is any process or action that can be repeated under the same conditions, and has more than one possible outcome, but you can't predict the exact result in advance.

There are two types:

1. Discrete Random Variable
2. Continuous Random Variable

1. Discrete Random Variable: It takes countable values such as, 0, 1, 2, 3, ...

Examples:

- Number of spam emails received in a day: 0, 1, 2, 3...
- Outcome of a dice roll: 1 to 6
- Number of likes on a social media post

2. Continuous Random Variable: It takes any value within a range including fractions. It can have infinite possible values.

Examples:

- Temperature in a room: 22.3°C, 22.31°C, etc.
- Weight of a person
- Confidence score of an AI prediction (0.0 to 1.0)

3.5 Probability Distribution

A Probability Distribution tells us how likely different values of a random variable are. Think of it as a map of probabilities across all possible outcomes.

A. For Discrete Random Variables: It uses a Probability Mass Function (PMF). Assigns probability to each specific value

Example:

Let X = number of heads in 2-coin tosses Values: 0, 1, 2

X	**P(X)**
0	0.25
1	0.50
2	0.25

B. For Continuous Random Variables: It uses a Probability Density Function (PDF). The probability of a single point is 0, but the area under a curve (interval) matters.

Example:

The height of people follows a normal distribution. Mean = 170 cm, SD = 10 cm

PDF gives the bell-shaped curve, and we find the probability of height within a range like: P(160 < X < 180)

3.5 Common Probability Distributions

The following table shows commonly used distributions. Figure 2 shows shapes of commonly used distributions.

Distribution	**Type**	**Use in AI**
Bernoulli	Discrete	Binary classification (0 or 1)
Binomial	Discrete	Repeated binary trials (e.g., flipping coins)
Poisson	Discrete	Number of events (e.g., clicks per second)
Uniform	Both	Equal probability (used in random initialization)
Normal (Gaussian)	Continuous	Natural data patterns, used in regression, loss functions
Exponential	Continuous	Time between events (e.g., time until failure)

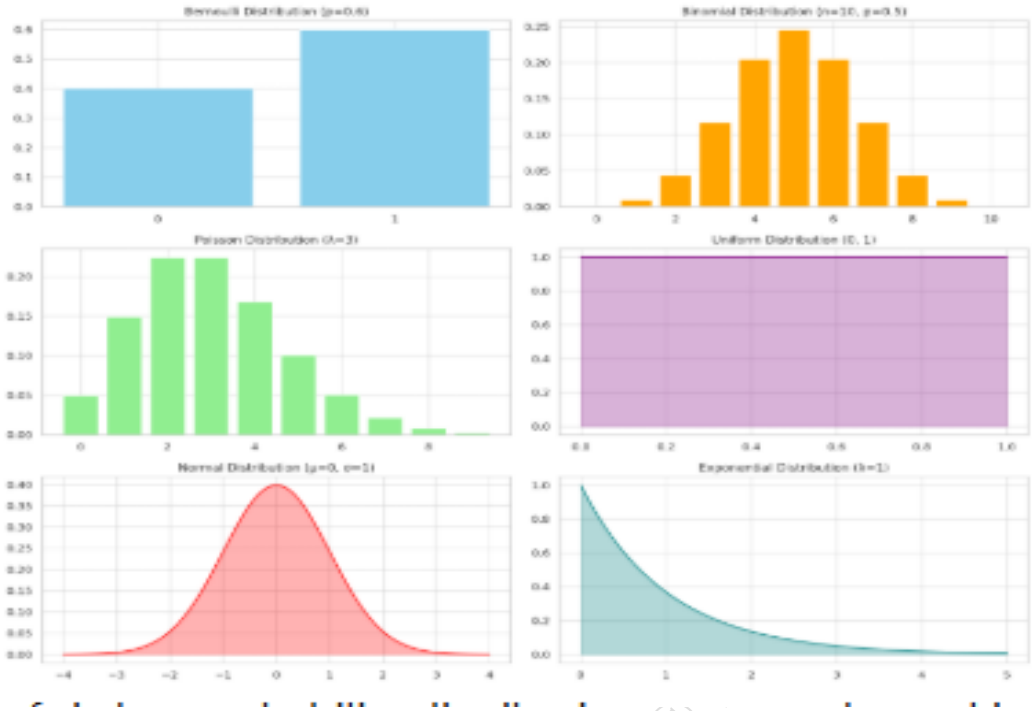


Fig. 3.2: Shapes of Probability Distributions

AI Use of Random Variables and Distributions

AI Application	Random Variable	Distribution
Spam detection	Words in email	Bernoulli / Multinomial
Image classification	Pixel intensity	Normal
Chatbot reply timing	Response delay	Exponential
Neural network weight init	Weights	Uniform / Normal
Forecasting	Predicted value	Gaussian or custom

Expectation (Expected Value)

The expected value is the average or mean outcome you'd expect if you repeated an experiment many times.

Formula (Discrete Random Variable):

$$E[X] = \sum_i x_i \cdot P(x_i)$$

Example:

Let X be the outcome of rolling a fair 6-sided die.

E[X] = (1+2+3+4+5+6)/6 = 3.5

Even though you never roll a 3.5, it is the theoretical average.

Use in AI:

1. Calculating expected loss
2. Understanding model predictions
3. Designing utility functions in decision-making

Variance

Variance measures how spread out the values of a random variable are from the mean.

Formula:

$$Var(X) = E[(X - E[X])^2]$$

Example (for the die):

$$E[X] = 3.5, \quad Var(X) = \frac{(1-3.5)^2 + ... + (6-3.5)^2}{6} = 2.9167$$

Low Variance means data is tightly clustered around the mean. High Variance means data is more spread out.

Use in AI:

1. Understanding model uncertainty
2. Feature scaling (e.g., normalization)
3. Regularization to reduce over fitting

Covariance

Covariance shows how two variables change together.

Formula:

$$Cov(X, Y) = E[(X - E[X])(Y - E[Y])]$$

Example:

1. If X = number of hours studied
2. Y = exam score

Then:

1. Positive covariance means more study, higher scores.
2. Negative covariance means more study, lower scores which is unusual.
3. Zero covariance means no relationship.

Use in AI:

1. Important in Principal Component Analysis (PCA)
2. Understanding relationships between features
3. Input for covariance matrices in multivariate statistics

Table:

Concept	**Measures**	**Output**	**AI Use**
Expectation	Central value	Mean	Model averaging, predictions
Variance	Spread from mean	Scalar	Error analysis, regularization
Covariance	Relationship between two vars	Positive/Negative/Zero	Feature selection, PCA

Practical Activities in Python

1. Theoretical Probability – Dice Roll

Activity: Compare theoretical vs experimental probability.

AI Use: Forms the foundation for Bayes' theorem, Markov chains.

Python Program:

```
import random
# Simulate rolling a fair die
trials = 10000
count_4 = 0
for _ in range(trials)
    roll = random.randint(1, 6)
    if roll == 4:
        count_4 += 1
# Theoretical probability = 1/6 ≈ 0.1666
experimental_prob = count_4 / trials
print("Theoretical Probability of 4:", 1/6)
print("Experimental Probability of 4:", experimental_prob)
```

Sample Output:

Theoretical Probability of 4: 0.16666666666666666

Experimental Probability of 4: 0.1679

2. Empirical Probability – Spam Detection Example

Activity: **Calculate spam probability from dataset.**

AI Use: Spam filters, sentiment analysis, recommender systems.

Python Program:

```
# Dataset: Emails with word "offer"
total_emails_with_offer = 200
spam_emails_with_offer = 150
empirical_prob = spam_emails_with_offer / total_emails_with_offer
print("Empirical Probability(email is spam | 'offer') =", empirical_prob)
```

Sample Output:

```
Empirical Probability(email is spam | 'offer') = 0.75
```

3. Conditional Probability – Medical Diagnosis Example

Activity: Students can try with different values.

AI Use: Medical AI, spam filtering, voice recognition.

Python Program:

```
# Suppose from hospital records:
# P(fever) = 0.3, P(flu) = 0.1, P(fever|flu) = 0.8
P_flu = 0.1
P_fever = 0.3
```

```
P_fever_given_flu = 0.8
# Conditional probability P(flu|fever) using Bayes
P_flu_given_fever = (P_fever_given_flu * P_flu) / P_fever
print("Probability(flu | fever) =", P_flu_given_fever)
```

Sample Output:

```
Probability(flu | fever) = 0.26666666666666666
```

4. Bayes' Theorem – Spam Email

Activity: Change dataset to see effect.
AI Use: Naive Bayes classifiers in spam filtering.

Python Program:

```
# Historical data:
P_spam = 0.3
P_word_given_spam = 0.6   # "win" appears in spam
P_word_given_notspam = 0.1
P_notspam = 1 - P_spam

# Total probability of word "win"
P_word = (P_word_given_spam * P_spam) + (P_word_given_notspam * P_notspam)

# Bayes: P(spam | word)
P_spam_given_word = (P_word_given_spam * P_spam) / P_word
print("Probability(email is spam | contains 'win') =", P_spam_given_word)
```

Sample Output:

Probability (email is spam | contains 'win') = 0.6428571428571429

5. Random Variables – Dice Simulation

Activity: Show probability distribution of random variables.
AI Use: Feature distribution, word counts in NLP.

Python Program:

```
import numpy as np
# Simulate dice outcomes
trials = 10000
outcomes = np.random.randint(1, 7, size=trials)

# Discrete random variable distribution
unique, counts = np.unique(outcomes, return_counts=True)
prob_dist = counts / trials

print("Dice Outcomes Probability Distribution:")
```

```
for u, p in zip(unique, prob_dist):
    print(f"P({u}) = {p:.2f}")
```

Sample Output:

```
Dice Outcomes Probability Distribution:
P(1) = 0.17
P(2) = 0.16
P(3) = 0.17
P(4) = 0.16
P(5) = 0.17
P(6) = 0.17
```

6. Normal Distribution – AI Confidence Scores

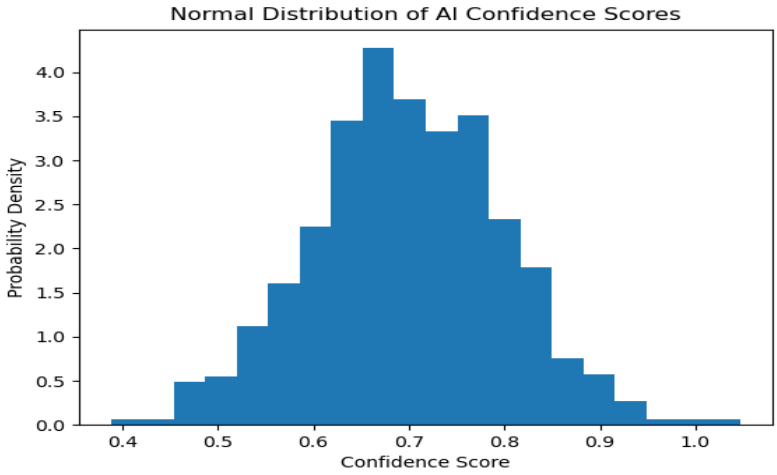
Activity: Plot and observe bell-shaped curve.
AI Use: Regression, loss functions, error analysis.

Python Program:

```
import matplotlib.pyplot as plt
# Simulate AI confidence scores ~ Normal(0.7, 0.1)
mean, std_dev = 0.7, 0.1
confidence_scores = np.random.normal(mean, std_dev, 1000)
plt.hist(confidence_scores, bins=20, density=True)
plt.title("Normal Distribution of AI Confidence Scores")
plt.xlabel("Confidence Score")
plt.ylabel("Probability Density")
plt.show()
```

Sample Output:

```
A bell-shaped curve centered around 0.7 appears.
X-axis: Confidence scores (≈ 0.4 to 1.0)
Y-axis: Probability density
```



7. Expectation and Variance – Dice Example

Activity: Compare with simulation results.

AI Use: Expected loss, model uncertainty, feature scaling

Python Program:

```
# Outcomes of fair die
outcomes = [1,2,3,4,5,6]
prob = [1/6] * 6
# Expectation
expectation = sum(o * p for o, p in zip(outcomes, prob))
# Variance
variance = sum(((o - expectation) ** 2) * p for o, p in zip(outcomes, prob))
print("Expected Value (Die Roll):", expectation)
print("Variance (Die Roll):", variance)
```

Sample Output:

```
Expected Value (Die Roll): 3.5
Variance (Die Roll): 2.9166666666666665
```

8. Covariance – Study Hours vs Exam Score

Activity: See positive covariance.

AI Use: PCA, feature selection, multivariate statistics.

Python Program

```
import numpy as np
# Example dataset
study_hours = np.array([2, 3, 4, 5, 6])
exam_scores = np.array([50, 60, 65, 70, 85])
# Covariance
cov_matrix = np.cov(study_hours, exam_scores)
print("Covariance Matrix:\n", cov_matrix)
```

Sample Output:

```
Covariance Matrix:
[[ 2.5  20. ]
 [20.  166. ]]
Positive covariance (20) means more study hours → higher exam scores.
```


Summary

Probability and Statistics help AI systems make decisions under uncertainty. Probability is used to predict the likelihood of events. For example, an AI model can calculate the probability that an email is spam or not spam.

Statistics helps in collecting, analysing, and interpreting data. Concepts like mean, median, variance, and standard deviation help summarize data, while probability distributions help model real-world situations. In AI, these concepts are used in tasks like spam detection, recommendation sys-

tems, medical diagnosis, and prediction models. Probability and Statistics enable AI systems to make accurate predictions and data-driven decisions.

Check Your Progress

A. Multiple Choice Questions

1. The probability of getting a head in a fair coin toss is (a) 0 (b) 0.25 (c) 0.5 (d) 1
2. Which of the following measures central tendency? (a) Mean (b) Standard Deviation (c) Variance (d) Range
3. Which probability rule is used when two events must both occur? (a) Addition rule b) Multiplication rule (c) Bayes' theorem (d) Conditional probability
4. The measure e of spread in data is given by (a) Mode (b) Median (c) Variance d) Frequency
5. Bayes' theorem is used for (a) Prediction (b) Probability updating (c) Sorting data d) Sampling

B. Fill in the Blanks

1. Probability values always lie between _____ and _____.
2. The most frequently occurring value in a dataset is called the _____.
3. Variance measures how far data points are from the _____.
4. _____ probability measures the likelihood of one event given another.
5. Standard deviation is the square root of _____.

C. True or False

1. Mean, median, and mode are measures of central tendency.
2. The probability of an impossible event is 1.
3. Variance can never be negative.
4. Bayes' theorem relates prior and posterior probabilities.
5. Standard deviation is always greater than variance.

D. Short Answer

1. Define probability with an example.
2. What is the difference between mean and median?
3. Explain the importance of variance in data analysis.
4. State Bayes' theorem in simple words.
5. Give one real-life example of probability use in AI.

Session 4. Discrete Mathematics for AI

4.0 Introduction

Discrete mathematics provides the logic and structures behind AI systems. Topics such as propositional and predicate logic, Boolean algebra, sets, functions, combinatorics, and graph theory form the basis for reasoning, data structures, and algorithms. These ideas support knowledge representation, search algorithms, and decision-making in AI.

Imagine a smart city crime investigation scenario, an AI-powered system can utilize principles of Discrete Mathematics to identify suspects in a series of connected robberies. It applies propositional and predicate logic to infer potential involvement based on presence and past records, uses sets to filter individuals with overlapping characteristics, and computes risk levels as functions.

A graph-based communication network uncovers hidden connections among suspects, while combinatorics helps analyse possible team formations and movement patterns. Finally, recursive algorithms can trace suspects' movements across locations. By combining these discrete math tools, the AI system intelligently narrows down the list of suspects with high confidence, enabling quicker and more accurate investigations. Figure 4.1 shows a smart crime investigation system.

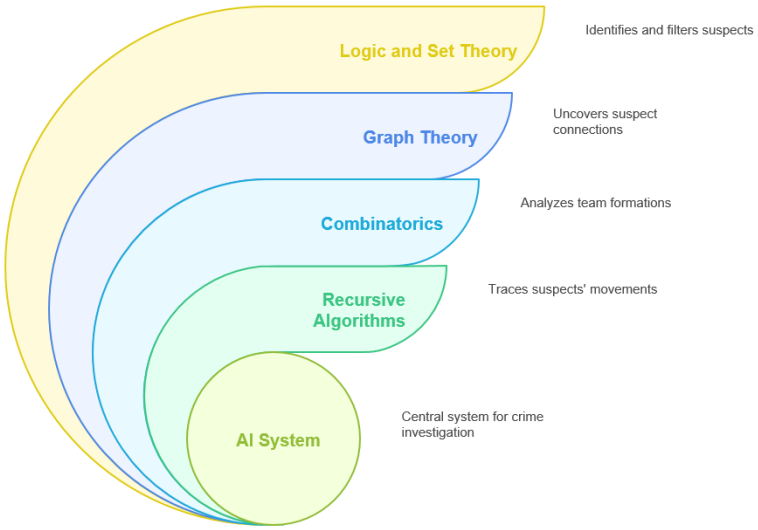


Fig. 4.1: Smart Crime Investigation System

Discrete Mathematics forms the theoretical foundation for many concepts in Artificial Intelligence (AI). It plays a critical role in designing algorithms, modelling knowledge, performing reasoning, and constructing intelligent systems. In this session we will discuss Discrete Mathematics.

Propositional Logic

Propositional logic is also called propositional calculus. It deals with statements (propositions) that are either true or false and the logical connectives between them such as, AND, OR, NOT and IMPLIES.

Relevance to AI:

- *Knowledge Representation:* AI systems represent facts about the world using logical statements.

- *Automated Reasoning:* Logical inference engines use rules of propositional logic to deduce new facts from known information.
- *Expert Systems:* Rule-based systems apply propositional logic to evaluate conditions and derive conclusions.

Example in AI:

If it is raining (R) → The road is wet (W)

R → W

If R is known to be true, AI infers W is true.

Model Logic Gates and Logical Expressions

Logic gates are fundamental building blocks of digital electronics. They perform basic logical functions that are essential for digital circuits. The behavior of logic gates can be modelled using Boolean algebra.

Common Logic Gates:

A logic gate is a logic circuit with one or more inputs and one output. It is used to build logical circuits. AND, OR and NOT are basic gates. By making combinations of basic gates we can construct derived gates such as NAND, NOR, XOR and XNOR. The following table and Figure 4.2 show commonly used logic gates along with symbol, expression and truth table.

Gate	Symbol	Boolean Expression	Truth Table
AND	A ∧ B	A AND B	1 only if both inputs are 1
OR	A ∨ B	A OR B	1 if at least one input is 1
NOT	¬A	NOT A	Inverts the input
NAND	¬(A ∧ B)	NOT AND	Inverse of AND
NOR	¬(A ∨ B)	NOT OR	Inverse of OR
XOR	A ⊕ B	A XOR B	1 if inputs are different
XNOR	¬(A ⊕ B)	NOT XOR	1 if inputs are the same

Logic Gates Symbols

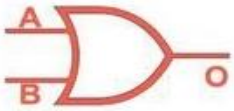
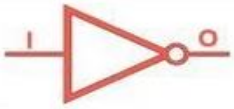
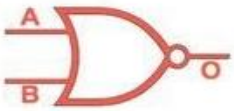
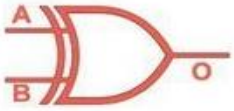
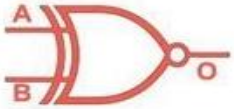
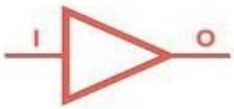
Gate	Symbol	Operator	Truth Table
AND		$O = A \cdot B$	Input A / Input B / Output 0 / 0 / 0 0 / 1 / 0 1 / 0 / 0 1 / 1 / 1
OR		$O = A + B$	Input A / Input B / Output 0 / 0 / 0 0 / 1 / 1 1 / 0 / 1 1 / 1 / 1
NOT		$O = \overline{I}$	Input / Output 0 / 1 1 / 0
NAND		$O = \overline{A \cdot B}$	Input A / Input B / Output 0 / 0 / 1 0 / 1 / 1 1 / 0 / 1 1 / 1 / 0
NOR		$O = \overline{A + B}$	Input A / Input B / Output 0 / 0 / 1 0 / 1 / 0 1 / 0 / 0 1 / 1 / 0
XOR		$O = A \oplus B$	Input A / Input B / Output 0 / 0 / 0 0 / 1 / 1 1 / 0 / 1 1 / 1 / 0
XNOR		$O = \overline{A \oplus B}$	Input A / Input B / Output 0 / 0 / 1 0 / 1 / 0 1 / 0 / 0 1 / 1 / 1
Buffer		$I = O$	Input / Output 0 / 0 1 / 1

Fig. 4.2: Logic Gates (Redraw)

Example Logical Expression:

Given:

A = 1, B = 0, C = 1

Expression: (A AND B) OR (NOT C)

Evaluation:

(A ∧ B) = 0, ¬C = 0, so result = 0 OR 0 = 0

Use in AI: Logic expressions are crucial in AI for decision-making, rule-based systems, and modeling knowledge using propositional or predicate logic.

Predicate Logic (First Order Logic - FOL)

Predicate logic extends propositional logic by using predicates and quantifiers such as, ∀ for all, ∃ there exists. It allows reasoning with variables and relations.

Relevance to AI:

a) *Semantic Web and Ontologies:* AI uses FOL to model relationships and classify entities.

b) *Natural Language Understanding:* Mapping sentences to FOL expressions enables AI to understand and reason with language.

c) *Planning and Robotics:* FOL is used in STRIPS and other AI planning systems to describe actions, goals, and states.

d) *Inference Engines:* Systems like Prolog are based on predicate logic and are used in symbolic AI.

Example in AI:

∀x (Human(x) → Mortal(x))

Human(Socrates) → Mortal(Socrates)

Sets

Sets are collections of distinct objects. Basic operations include union, intersection, difference, and Cartesian product.

Relevance to AI:

a) *Data Representation:* AI represents knowledge, states, actions, and features using sets.

b) *Search Spaces:* In AI search algorithms, the state space is a set of all possible states.

c) *Machine Learning:* Training and testing data are treated as sets of labeled/unlabeled examples.

Example in AI:

1. Set of symptoms = {fever, cough, fatigue}
2. AI diagnosis systems use this set to determine likely diseases.

Basic Set Operations

Let's define two sets:

1. A = {1, 2, 3, 4}
2. B = {3, 4, 5, 6}

1. Union (A ∪ B): It is a set of all elements that are in A, B, or both.

A ∪ B = {1, 2, 3, 4, 5, 6}

2. Intersection (A ∩ B): It is a set of elements common to both A and B.

A ∩ B = {3, 4}

3. Difference (A − B): It is a set of elements in A that are not in B.

A − B = {1, 2}

B − A = {5, 6}

4. Symmetric Difference (A △ B): It is a set of elements in A or B but not both.

A △ B = (A ∪ B) − (A ∩ B) = {1, 2, 5, 6}

5. Subset (A ⊆ B): If all elements of A are also in B then A is subset of B. Here, A ⊆ B? → No, because 1 and 2 are not in B. Figure 3 shows set operations.

Let C = {3, 4}

C ⊆ A? → Yes

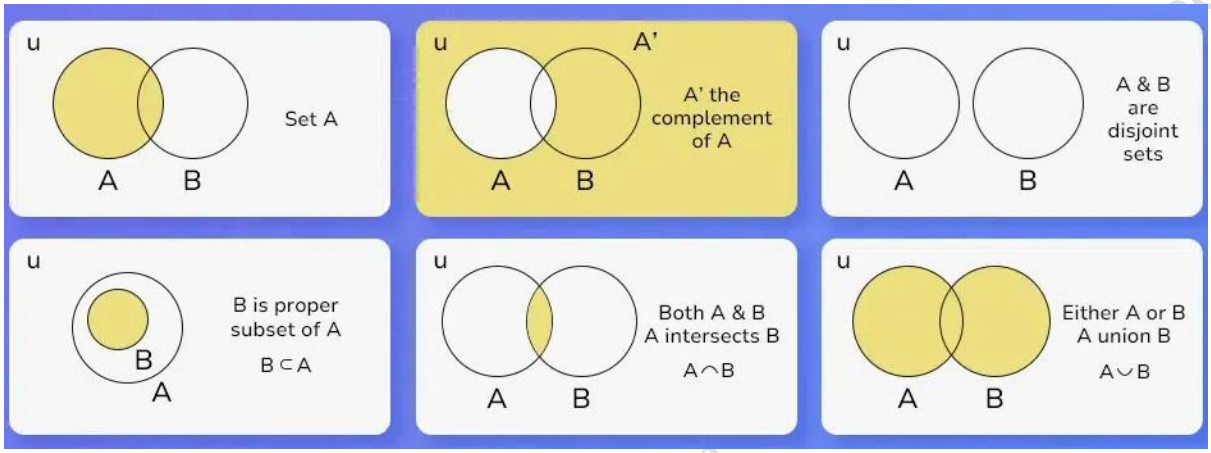


Fig 3: Set Operations (Redraw)

6. Power Set (P(A)): It is a set of all subsets of A.

P(A) = { {}, {1}, {2}, {3}, {4}, {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}, {1,2,3}, {1,2,4}, {1,3,4}, {2,3,4}, {1,2,3,4} }

(2^4 = 16 subsets)

7. Cartesian Product (A × B): It is a set of all ordered pairs (a, b) where a ∈ A and b ∈ B.

A × B =
{(1,3), (1,4), (1,5), (1,6),
(2,3), (2,4), (2,5), (2,6),
(3,3), (3,4), (3,5), (3,6),
(4,3), (4,4), (4,5), (4,6)}

Total = 4 × 4 = 16 pairs

Application in AI

1. Union and Intersection: Used in feature selection, fuzzy logic, and knowledge merging.
2. Difference: Detecting unique or novel data in datasets.
3. Power Set: Used in combinatorial optimization and rule mining.
4. Cartesian Product: Used to model state spaces or input combinations.

Functions

A function maps element from one set (domain) to another (codomain). Functions may be injective, surjective, or bijective.

A function is a relation that assigns exactly one output to each input from a set. If f: A→B, then for every a∈A, there is a unique b∈B such that f(a) = b.

Relevance to AI:

- *Neural Networks:* Functions model the transformation from inputs to outputs via layers.
- *Search and Optimization:* Objective functions guide search and learning algorithms.
- *Heuristic Functions:* In A* algorithm, a heuristic function estimates the cost from the current node to the goal.

Example in AI:

f(x) = predicted class of input x in a classifier.

Types of Functions

1. One-to-One Function (Injective): Each element of the domain maps to a unique element in the codomain. It is used in cryptography, hashing.

Example:

Let f(x) = 2x, with domain {1, 2, 3}

Then f = {(1,2),(2,4),(3,6)} — all outputs are unique.

2. Onto Function (Surjective): Every element of the codomain is mapped by at least one element of the domain. It is important in logic programming and range coverage problems.

Example:

Let $f(x) = x^2$ with domain = {–2, –1, 0, 1, 2} and codomain = {0, 1, 4}
Then f = {(−2,4),(−1,1),(0,0),(1,1),(2,4)} — codomain is fully covered.

3. One-to-One and onto (Bijective): It is a function that is both injective and surjective — a perfect pairing between domain and codomain. It is used in AI where reversible mappings are needed (e.g., encoder-decoder models).

Example:
Let f(x) = x+1, Domain = {1, 2, 3}, Codomain = {2, 3, 4}
Then f = {(1,2),(2,3),(3,4)} — unique and full mapping.

4. Many-to-One Function: Here, multiple inputs map to the same output. It is common in classification tasks where multiple states belong to one class. Figure 4 shows types of functions.

Example:
$f(x) = x^2$, Domain = {–2, –1, 0, 1, 2}
Then f = {(−2,4),(−1,1),(0,0),(1,1),(2,4)} — different x-values yield the same y.

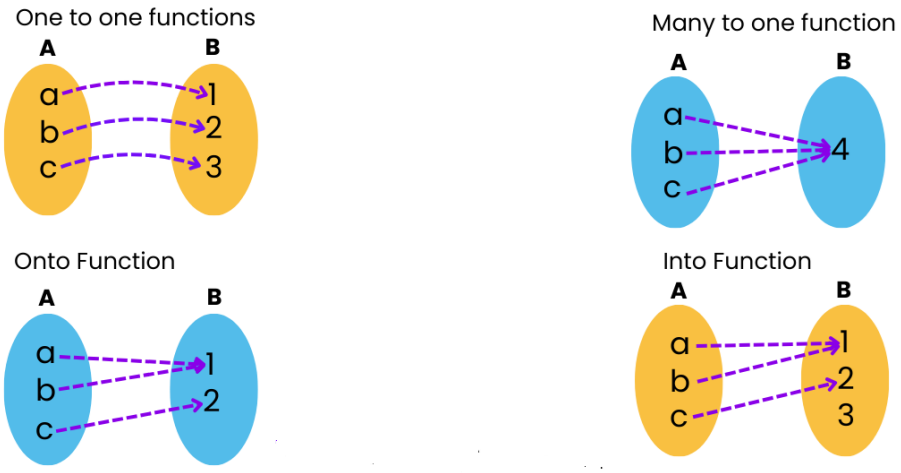


Fig. 4.4: Types of Functions (Redraw)

5. Constant Function: Here, every input maps to the same output. It models bias terms in ML and zero-variance scenarios.

Example:

f(x) = 7, for all x in the domain

Then f = {(x, 7) for all x}

6. Identity Function: Here, each input maps to itself. It is used in AI pipelines where transformations are bypassed.

Example: f(x) = x

7. Modulus Function: It gives the absolute value of the input. It is common in loss functions like L1 loss in ML.

Example: If f(x) = |x|, so f(−3) = 3, f(2) = 2

8. Step Function (Greatest Integer Function): Also called floor function. Returns the greatest integer ≤ x. It is used in thresholding digital circuits.

Example:

f(x) = ⌊x⌋, f(2.7) = 2, f(−1.2) = −2

Use of Functions in AI

Type	**Use Case**
Injective	Encryption, Mapping States
Surjective	Logic Programming
Bijective	Invertible Functions
Constant	Baseline models
Identity	No transformation layers
Step	Threshold logic
Modulus	Loss functions in AI

Graphs

Graphs consist of vertices (nodes) and edges (connections). They can be directed or undirected, weighted or unweighted.

Relevance to AI:

- *Search Algorithms:* AI uses graphs in pathfinding (A*, Dijkstra's, BFS, DFS).
- *Knowledge Graphs:* AI systems use graphs to represent and reason over structured knowledge (e.g., Google Knowledge Graph).
- *Social Networks, Recommendation Systems:* AI models relationships using graphs.
- *Bayesian Networks:* Probabilistic graphical models represent conditional dependencies among variables.

Example in AI:

A graph of cities and roads used by an AI agent in GPS-based route planning.

Graph Basics: Nodes, Edges, and Adjacency Matrices

1. Nodes (Vertices)

- Definition: Nodes (or vertices) are the fundamental units of a graph that represent entities such as people, places, devices, or states.
- Notation: Typically represented as V = {v1,v2,v3,...}
- Example: In a social network, each person is a node.

In AI, nodes can represent states in a search problem or neurons in a neural network.

2. Edges (Links)

- Definition: Edges are connections or relationships between two nodes.
- Types:
  - Undirected Edge: No direction — A↔B
  - Directed Edge (Arc): One-way — A→B
  - Weighted Edge: Carries a cost or value, e.g., distance or time.
- Notation: E = {(vi,vj)}

In AI, edges can represent transitions, paths, or dependencies.

3. Adjacency Matrix

- Definition: A 2D matrix used to represent a graph. If the graph has n nodes, the matrix is of size n × n.
- Usage: Indicates whether an edge exists between any two nodes. Figure 5 shows graph and adjacency matrix.

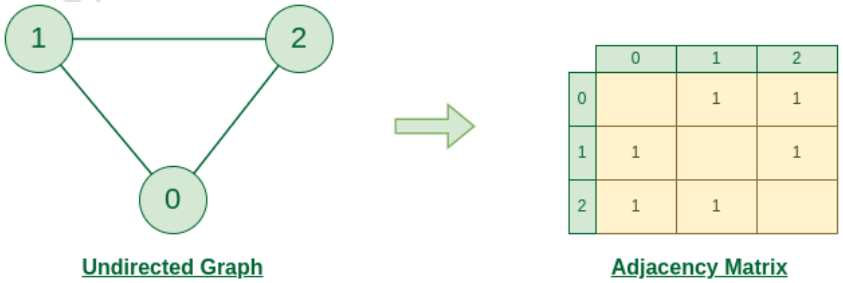


Fig. 4.5: Graph and Adjacency Matrix (Redraw)

Example: Graph with 3 Nodes

Let the nodes be:
A = 0, B = 1, C = 2

Let the edges be:
A → B, B → C, A → C

Adjacency Matrix (Directed Graph)

	A(0)	B(1)	C(2)
A	0	1	1
B	0	0	1
C	0	0	0

1. A to B = 1 (edge exists)
2. A to C = 1
3. Others = 0 (no edge)

For an Undirected Graph, the matrix is symmetric.

Weighted Adjacency Matrix: If edges carry weights, we replace 1s with actual values:

	A	B	C
A	0	2	5
B	0	0	3
C	0	0	0

Here:

- A→B has weight 2
- A→C has weight 5
- B→C has weight 3

Applications in AI

Concept	**AI Application**
Nodes	States, Concepts, Devices
Edges	Transitions, Relations, Flows
Adjacency Matrix	Representing Search Spaces, Knowledge Graphs, Neural Networks

Solve Graph Search Problems using NetworkX in Python

Graph traversal is a fundamental AI problem-solving technique, often used in pathfinding, state-space search, and knowledge representation.

Graph Search Algorithms:

- *DFS (Depth-First Search):* Explores as far as possible along a branch before backtracking.
- *BFS (Breadth-First Search):* Explores all neighbors before going deeper.

Python Example using NetworkX:

```
import network x as nx
# Create graph
G = nx.Graph()
edges = [('A', 'B'), ('A', 'C'), ('B', 'D'), ('C', 'E'), ('E', 'F')]
G.add_edges_from(edges)

# DFS
dfs_nodes = list(nx.dfs_preorder_nodes(G, source='A'))
print("DFS:", dfs_nodes)

# BFS
bfs_nodes = list(nx.bfs_tree(G, source='A'))
print("BFS:", bfs_nodes)
```

Output:

```
DFS: ['A', 'B', 'D', 'C', 'E', 'F']
BFS: ['A', 'B', 'C', 'D', 'E', 'F']
```

Use in AI:

It is used in planning, puzzle solving (e.g., 8-puzzle), route finding, and reasoning over knowledge graphs.

Combinatorics

Combinatorics deals with counting, arrangements, and combinations of elements in a set.

Relevance to AI:

- *Search Complexity:* Helps estimate the number of possible states or solutions.
- *Probability and Statistics:* Essential in probabilistic AI and ML.
- *Model Complexity:* Understanding hypothesis spaces in ML (e.g., number of decision trees of depth d).
- *Game Theory and Decision Making:* Combinatorics helps in evaluating strategies.

Example in AI:

Number of possible configurations of a Rubik's Cube = 43 quintillion. Combinatorics helps AI design efficient solvers.

Construct Decision Trees and Analyse Combinatorial Paths

A decision tree is a supervised machine learning algorithm used for classification and regression tasks. It works by splitting the dataset into subsets based on feature values, forming a tree-like structure where each internal node represents a decision on an attribute, each branch represents an outcome of the decision, and each leaf node represents a final class label or value.

The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from data features. Decision trees are easy to understand

and visualize, making them useful for explaining decisions and identifying key influencing variables.

Example Use Case: Weather-based decision tree to play cricket.

Outlook	Humidity	Windy	Play?
Sunny	High	False	No
Sunny	Normal	False	Yes
Overcast	High	True	Yes

Features: Outlook, Humidity, Windy
Target: Play?

Build Using Python (e.g., sklearn):

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# Sample Data
X = [[0, 1, 0], [0, 0, 0], [1, 1, 1]]
y = [0, 1, 1]  # 0 = No, 1 = Yes

# Train Model
clf = DecisionTreeClassifier()
clf = clf.fit(X, y)

# Plot Tree
tree.plot_tree(clf, feature_names=['Outlook', 'Humidity', 'Windy'])
```

Combinatorial Path Analysis: When analysing decision paths, every decision node leads to branches. The total number of possible paths depends on tree depth and splits per node.

Example: A binary decision tree of depth 3 can have up to $2^3 = 8$ unique decision paths.

Use in AI: It is used in classification models, expert systems, game playing (minimax trees), and explainable AI.

Recursion

Recursion refers to a process where a function calls itself to solve a problem by breaking it down into smaller sub-problems.

Relevance to AI:

- *Tree Traversal:* Recursive algorithms are used in decision trees, game trees (minimax), and parsing.
- *Search Algorithms:* Recursive depth-first search (DFS).
- *Backtracking:* Used in constraint satisfaction problems (Sudoku, N-Queens).

- *Divide-and-Conquer:* Core to many AI algorithms (e.g., Merge Sort in preprocessing, Quicksort in clustering).

Example in AI:

Recursive definition of the minimax algorithm in AI games.

Recursion Example: Factorial Function

A function that calls itself to solve smaller instances of the same problem is called recursion.

Factorial of a number n

```
n! = n × (n−1)! With base case: 0!=1
Python Code Example:
def factorial(n):
    if n == 0:
        return 1   # Base case
    else:
        return n * factorial(n - 1)  # Recursive call

print(factorial(5))  # Output: 120
```

Recursive Flow for factorial (5):

```
factorial(5)
= 5 * factorial(4)
= 5 * (4 * factorial(3))
= 5 * (4 * (3 * factorial(2)))
= 5 * (4 * (3 * (2 * factorial(1))))
= 5 * (4 * (3 * (2 * (1 * factorial(0)))))
= 5 * 4 * 3 * 2 * 1 * 1 = 120
```

Key Elements of Recursion:

- Base Case: Stops recursion — prevents infinite loop.
- Recursive Case: Function calls itself with smaller/simpler input.

Applications of Recursion in AI:

Area	**Example**
Search Algorithms	Depth-First Search (DFS)
Problem Solving	Tower of Hanoi, Backtracking
Natural Language Parsing	Grammar rule expansion
Tree Traversals	Inorder, Preorder, Postorder

Each concept in Discrete Mathematics provides foundational tools and language for AI. Whether it's modelling relationships using predicate logic, solving problems through recursion, or analysing state spaces with graphs and combinatorics, discrete mathematics is deeply interwoven into the theory and practice of AI. A strong grasp of these topics is essential for designing robust, intelligent systems.

Table:

Topic	Application in AI
Logic Gates & Expressions	Rule-based AI, knowledge representation
Graph Search (DFS/BFS)	Search algorithms, navigation, planning
Decision Trees & Paths	Machine learning, classification, decision analysis

Python Programs

1. Propositional Logic – Rule-Based Reasoning

AI Use: Models if-then rules in expert systems (medical diagnosis, recommendation engines).

Python Code:

```
# Simple propositional logic: If it rains, the road is wet.
R = True   # It is raining
W = False  # Road wet initially
if R:
    W = True

print("Raining:", R)
print("Road Wet:", W)
```

Sample Output:

```
Raining: True
Road Wet: True
```

2. Logic Gates – Boolean Operations

AI Use: Basic building blocks of neural networks and digital reasoning circuits.

Python Code:

```
# Logic gates using Python functions
def AND(a, b): return a and b
def OR(a, b): return a or b
def NOT(a): return not a
def XOR(a, b): return a ^ b

A, B = 1, 0
print("AND:", AND(A, B))
print("OR:", OR(A, B))
print("NOT A:", NOT(A))
print("XOR:", XOR(A, B))
```

Sample Output:

```
AND: 0
```

```
OR: 1
NOT A: False
XOR: 1
```

3. Predicate Logic – Knowledge Representation

AI Use: Used in semantic web, ontologies, and natural language understanding.

Python Code:

```
# Example: All humans are mortal, Socrates is a human → Socrates is mortal
humans = ["Socrates", "Plato", "Aristotle"]
mortal = {}
for h in humans:
    mortal[h] = True  # ∀x (Human(x) → Mortal(x))
print("Is Socrates mortal?", mortal["Socrates"])
```

Sample Output:

```
Is Socrates mortal? True
```

4. Sets – Operations in AI

AI Use: Search spaces, feature selection, knowledge merging.

Python Code:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
print("Union:", A | B)
print("Intersection:", A & B)
print("Difference A-B:", A - B)
print("Symmetric Difference:", A ^ B)
```

Sample Output:

```
Union: {1, 2, 3, 4, 5, 6}
Intersection: {3, 4}
Difference A-B: {1, 2}
Symmetric Difference: {1, 2, 5, 6}
```

5. Functions – Mapping Inputs to Outputs

AI Use: Neural networks model input-output mappings, loss functions

Python Code:

```
# f(x) = x^2
def f(x): return x**2
for x in [-2, -1, 0, 1, 2]:
```

```
    print(f"{x} → {f(x)}")
```

Sample Output:

```
-2 → 4
-1 → 1
0 → 0
1 → 1
2 → 4
```

6. Graphs – Search Algorithms (DFS, BFS)

AI Use: Pathfinding, planning, puzzle solving (e.g., Google Maps, 8-puzzle).

Python Code:

```
import networkx as nx
# Create graph
G = nx.Graph()
edges = [('A', 'B'), ('A', 'C'), ('B', 'D'), ('C', 'E')]
G.add_edges_from(edges)
# DFS and BFS
print("DFS:", list(nx.dfs_preorder_nodes(G, source='A')))
print("BFS:", list(nx.bfs_tree(G, source='A')))
```

Sample Output:

```
DFS: ['A', 'B', 'D', 'C', 'E']
BFS: ['A', 'B', 'C', 'D', 'E']
```

7. Combinatorics – Counting Possibilities

AI Use: Estimates state spaces, hypothesis sets, game tree complexity.

Python Code:

```
import itertools
items = ['A', 'B', 'C']
perms = list(itertools.permutations(items, 2))
print("Permutations of 2 items:", perms)
Sample Output:
Permutations of 2 items: [('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'C'),
('C', 'A'), ('C', 'B')]
```

8. Recursion – Factorial Example

AI Use: Recursive search in DFS, minimax algorithm, backtracking (Sudoku, N-Queens).

Python Code:

```
def factorial(n):
```

```
    if n == 0:
        return 1
    return n * factorial(n-1)
print("Factorial(5):", factorial(5))
```

Sample Output:

```
Factorial(5): 120
```

Summary

Discrete Mathematics provides the logical foundation for Artificial Intelligence. It deals with logic, sets, relations, functions, graphs, and combinatorics, which are essential for computer-based systems.

- Logic is used in decision-making and rule-based AI systems.
- Graph theory helps in representing networks like social media connections and search algorithms.
- Sets and relations help organize and structure data.
- Combinatorics is used to calculate possible outcomes in problem-solving and optimization tasks.

In AI, discrete mathematics supports areas such as search algorithms, knowledge representation, cybersecurity, and data structures. Thus, Discrete Mathematics helps AI systems think logically and solve complex problems efficiently.

Check Your Progress

A. Multiple Choice Questions

1. Which of the following is a set? (a) {1, 2, 3} (b) 123 (c) (1, 2) (d) 1 + 2
2. The number of subsets of a set with 3 elements is (a) 3 (b) 6 (c) 7 (d) 8
3. In logic, the statement “NOT True” is (a) True (b) False (c) Undefined (d) 1
4. Which data structure is based on nodes and edges? (a) Array (b) Graph (c) Stack (d) Queue
5. Which is used in AI for shortest path algorithms? (a) Graph theory (b) Calculus (c) Matrices (d) Probability

B. Fill in the Blanks

1. A collection of well-defined objects is called a _____.
2. A statement that is either true or false is called a _____.
3. In Boolean algebra, 1 represents _____ and 0 represents _____.
4. A _____ is a data structure made of nodes and edges.
5. The total number of subsets of a set with n elements is _____.

C. True or False

1. Sets can contain duplicate elements.
2. A proposition must be either true or false.
3. In Boolean algebra, AND operation is represented by multiplication.
4. Graphs are not useful in AI applications.
5. Logic is the foundation of reasoning in AI.

D. Short Answer Questions

1. Define a set with an example.
2. What is a proposition in logic?
3. Explain the role of Boolean algebra in AI.
4. Differentiate between a tree and a graph.
5. Give one AI application of graph theory.

Module 5. Fundamentals of Machine Learning

Module Overview

Machine Learning (ML) is one of the most fascinating and practical branches of Artificial Intelligence (AI). It enables computers to learn from data, recognize patterns, and make predictions without being explicitly programmed for each task. From smart refrigerators that remind families to buy milk, to voice assistants, self-driving cars, and medical diagnosis systems, ML is transforming the way we live and work.

A child was fascinated by his family's new refrigerator, as shown in Figure 5.1. It has a small screen, could track the items inside, and even reminded his mother when the milk was about to run out. One evening, he looked at the fridge, and it displayed:

"Milk is low. Based on your pattern, you'll need to buy some by tomorrow."

He was amazed. "How did it know that?" he asked his father.

His father smiled, "It's not magic—it's Machine Learning. The fridge has learned our habits over time. It knows what we store, how quickly we use things, and makes smart suggestions."

Fig. 5.0: Smart AI-Refrigerator

His curiosity was sparked. He started searching online and watching videos. Soon, he discovered that Machine Learning powers many things around him: phone assistants, online games, shopping suggestions, and even automatic photo tagging. That day, he decided he wanted to dive deeper into this fascinating field. By observing such real-life examples like Smart AI-Refrigerator, students will grasp how machines use data to learn, adapt, and assist humans in daily tasks.

In this Module, students will explore the fundamentals of Machine Learning (ML) and understand how it connects to Artificial Intelligence (AI).

Learning Outcomes

After completing this module, you will be able to:

- Explain the basic concept of Machine Learning (ML) and understand how it differs from traditional programming.

- Identify different types of Machine Learning such as Supervised Learning, Unsupervised Learning, and Reinforcement Learning.
- Understand the role of data in Machine Learning, including training data, testing data, features, and labels.
- Recognize ethical issues and responsibilities related to AI/ML such as bias, privacy, fairness, and transparency.
- Use no-code ML tools to build simple machine learning models and interpret results.
- Appreciate the real-world applications of ML in various domains such as healthcare, finance, and education.

Module Structure

Session 1. Introduction to Machine Learning

Session 2. Types of Machine Learning

Session 3. Data in Machine Learning

Session 4. Ethics and Responsibility in AI/ML

Session 5. Hands-On ML Without Code (Using Tools)

Session 1. Introduction to Machine Learning

1.0. Introduction

Machine Learning is like teaching a child through examples rather than giving strict instructions. For example, if you show a child many pictures of apples and bananas, the child gradually learns to recognize them without explaining the rules. Similarly, in ML, computers learn patterns from data. A common use case of ML is in the detection of diseases from medical records, where machines analyze symptoms and medical history to identify possible health issues. Just like a doctor uses past patient records to diagnose diseases, ML systems can analyze medical data to predict illnesses.

Fig. 1.1

This session introduces the basic idea of Machine Learning (ML) — teaching computers to learn from data instead of writing fixed rules. It explains the ML process (data collection, training, testing, and improvement) and its wide applications.

1.1 Machine Learning (ML)

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that empowers computers to learn from data and make decisions or predictions without requiring manual programming for every task. Unlike traditional programming, where developers write specific rules, ML leverages patterns within data to derive informed decisions. This generalization capability allows machines to effectively process new, unseen data.

ML depends on mathematical models and algorithms that enhance performance automatically through input data, eliminating the need for explicit programming for each individual task. Figure 1.1 shows the input data can come in diverse formats such as text, numbers, images, audio, or video. The machine analyzes this data to identify patterns, relationships, and hidden structures, mimicking how humans learn from experience.

Fig. 1.2: Machine Learning

1.1. Process of Machine Learning

Example – Classifying Fruit as Fresh or Rotten

To understand how machine learning works, let's explore a simple example where a machine is trained to identify whether a fruit is fresh (1) or rotten (0) based on its features.

Step 1. Data Collection

The machine is given a small dataset of fruits, along with labels showing whether each fruit is fresh or rotten as we can see in the Table 1.1 Dataset of fruits.

Table 1.1 Dataset of fruits.

Color	Weight (grams)	Smell	Label (Fresh or Rotten)
Green	150	Pleasant	1 (Fresh)
Brown	140	Foul	0 (Rotten)
Yellow	160	Pleasant	1 (Fresh)
Black	130	Foul	0 (Rotten)

Step 2. Feature Extraction

The machine identifies important features such as:

1. Color of the fruit
2. Weight
3. Smell quality (Pleasant or Foul)

These features help the model distinguish between fresh and rotten fruits.

Step 3. Model Training

Now, the machine uses an algorithm (such as Decision Trees or Logistic Regression) to learn from the training data. It begins to associate:

1. Green or yellow color + pleasant smell → Fresh
2. Brown or black color + foul smell → Rotten

Step 4. Testing and Prediction

Once trained, the machine can now classify new fruits.

New Fruit: Yellow, 155 grams, Pleasant smell

→ Machine identifies features → Predicts: Fresh (1)

New Fruit: Brown, 135 grams, Foul smell

→ Machine identifies features → Predicts: Rotten (0)

Step 5. Continuous Learning

If a user corrects a prediction (e.g., marks a "Fresh" fruit as "Rotten"), the machine updates its model. Over time, the more data it gets, the smarter and more accurate it becomes.

1.2. Importance and Benefits of Machine Learning

1. Data-Driven Decision Making: Machine Learning enables systems to make informed decisions by learning from data such as images, text, and historical records. Instead of following fixed rules, ML models identify patterns in data to generate accurate predictions and real-time decisions.

2. Automation of Tasks: ML automates both simple and complex tasks using algorithms like decision trees, neural networks, and support vector machines. Trained models can perform functions such as handwriting recognition or equipment monitoring, reducing manual effort and human error.

3. Increased Efficiency and Productivity: ML improves efficiency across industries. It powers recommendation systems, speech recognition, medical diagnostics, and autonomous vehicles. It also supports fraud detection, chatbots, and personalized ads, helping organizations streamline operations and serve users better.

4. Better Accuracy and Adaptability: ML systems improve over time by learning from new data. This continuous learning enhances their ability to adapt and make more accurate decisions, which is especially useful in sensitive fields like finance, security, and healthcare.

5. Scalability and Societal Impact: AI and ML scale to serve millions, making them ideal for large applications. From smart assistants to fraud detection and autonomous transport, these technologies are changing how we live, work, and interact—reshaping industries and society.

1.3. Workflow of Machine Learning

Workflow of Machine Learning involves a systematic sequence of steps aimed at building, testing, and implementing a machine learning model effectively.

Step 1. Problem Definition – The first step is to precisely define the problem to be addressed using a machine learning approach. Determine whether the goal is classification, regression, or clustering, and establish key performance indicators such as accuracy, precision, and recall. Also, decide on the format of the output—whether it will be a specific label or a probability score.

Step 2. Data Collection – Gather appropriate and sufficient data from various sources like databases, sensors, or web scraping tools. It is essential that the collected dataset includes meaningful features and is both diverse and representative of the problem domain.

Step 3. Data Pre-processing – Prepare the raw data by cleaning and transforming it. This includes handling missing values, removing noise, normalizing data, and formatting it properly for analysis and model input.

Step 4. Data Analysis – Explore and understand the data through descriptive statistics and visualization. Identify trends, correlations, and patterns that can guide the modelling process.

Step 5. Model Selection – Choose the most suitable machine learning algorithm based on the problem type and data characteristics. This could involve models like Support Vector Machines, Neural Networks, or K-Means clustering, among others.

Step 6. Model Training – Use the prepared data to train the selected model. During this process, the model learns patterns and relationships that will help it make predictions on new data.

Step 7. Model Evaluation – Evaluate the model's effectiveness by testing it on unseen data and measuring its performance using metrics like accuracy, precision, recall, and F1-score. If the model does not meet the training goals, then tune the parameters and re-train the model.

Step 8. Model Deployment – After successful evaluation, deploy the trained model into a real-world system or application where it can be used for actual predictions or decision-making.

Note: Continuously monitor the deployed model's performance in the real environment. Update or retrain the model periodically to ensure it adapts to new data and maintains its accuracy over time.

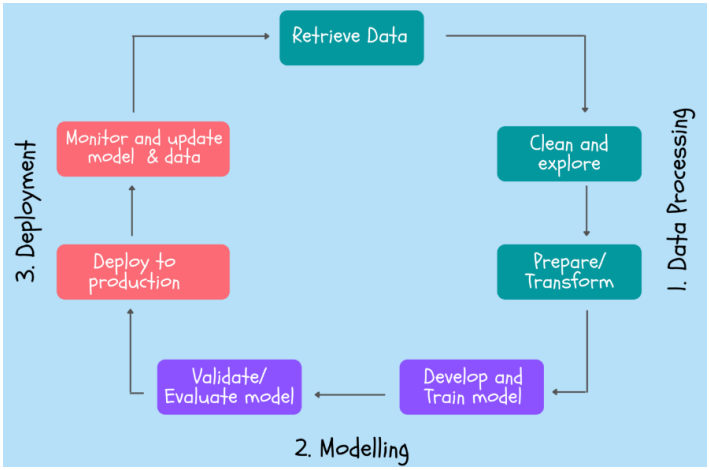


Fig. 1.3: Machine Learning Workflow

1.4. Applications of Machine Learning

Machine Learning (ML) finds applications across a wide range of industries and domains. Some of the key areas where ML is being effectively implemented are shown in Figure 1.3 and outlined below:

1. **Healthcare** – ML is applied in diagnosing diseases, customizing treatment plans, predicting patient outcomes, discovering new drugs, and analyzing medical images for faster and more accurate interpretations.
2. **Banking and Finance** – In the financial sector, ML helps detect fraudulent transactions, assess creditworthiness, perform algorithmic trading, manage financial risk, and automate customer support through intelligent systems.
3. **Retail and E-Commerce** – ML enhances user experience by recommending products based on browsing patterns and preferences, as seen on platforms like Amazon and Netflix. It also supports dynamic pricing strategies and enables customer segmentation based on shopping behavior.
4. **Transportation** – Machine learning powers self-driving vehicles by helping them make real-time decisions. It is also used to predict traffic patterns and optimize route planning.
5. **Manufacturing** – ML assists in identifying defects during the production process and enhances supply chain efficiency by forecasting demand and streamlining logistics.
6. **Energy Sector** – In the energy industry, ML supports smart grid management and predicts power generation from renewable sources like solar and wind, thereby improving efficiency and sustainability.
7. **Entertainment** – ML algorithms recommend movies, personalize gaming experiences, and aid in intelligent content creation for users based on their interests and behavior.
8. **Security** – In the area of cybersecurity, ML is used for detecting anomalies, enabling biometric verification, and preventing cyber-attacks through behavior analysis.
9. **Natural Language Processing (NLP)** – ML drives language translation tools, sentiment analysis, and voice-powered virtual assistants such as Siri and Google Assistant.
10. **Environmental Applications** – ML contributes to climate prediction models, supports wildlife monitoring and conservation efforts, and improves waste management systems.
11. **Human Resources** – In HR, ML helps in candidate screening, analyzing employee performance, predicting attrition, and optimizing workforce productivity.

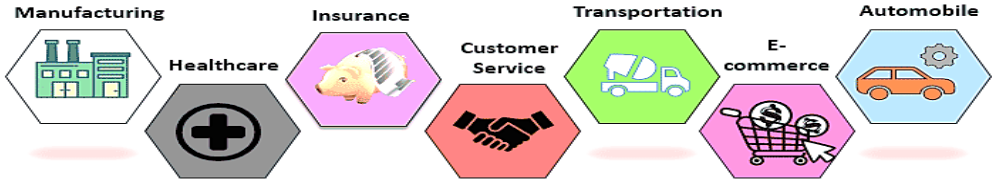


Fig. 1.4: Applications of Machine Learning

1.5. Real-life Examples of ML

Machine Learning has become a core technology behind many intelligent systems that simplify our daily lives and business operations. Here are some key real-world applications:

1. **Email Spam Detection:** ML algorithms analyze patterns in emails to distinguish spam from legitimate messages. Over time, these models adapt to new types of spam.
2. **Product Recommendations:** Platforms like Amazon and Netflix use ML to understand user preferences and suggest relevant products or content, increasing user engagement and sales.
3. **Voice Assistants:** Siri, Google Assistant, and Alexa use ML to recognize speech, interpret commands, and respond naturally, learning from interactions to improve over time.
4. **Self-Driving Cars:** Autonomous vehicles use ML for object detection, lane tracking, and decision-making in real-time environments, improving with every mile driven.
5. **Fraud Detection:** Banks and credit card companies use ML to flag unusual transactions by identifying anomalies in user behavior that may indicate fraud.
6. **Medical Diagnostics:** ML aids in diagnosing diseases by analyzing medical images and patient history, often providing insights beyond human capability.
7. **Social Media Feeds:** ML curates content in platforms like Facebook and Instagram, deciding what to show users based on their behavior and interests.

1.6. Difference Between AI, ML, DL

Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are interrelated fields, often used interchangeably, but they differ in scope and functionality. Table 1.2 shows the key differences between AI, ML, and DL with aspects.

AI is the broadest concept, aiming to simulate human intelligence.

ML is a subset of AI that enables machines to learn from data.

DL is a subset of ML that uses deep neural networks for complex tasks.

Table 1.2: Key differences between AI, ML, and DL with aspects

Aspect	Artificial Intelligence (AI)	Machine Learning (ML)	Deep Learning (DL)
Definition	Simulates human intelligence in machines	Enables machines to learn from data	Uses neural networks to learn from large volumes of data
Scope	Broad field including ML, DL, robotics, NLP, etc.	Subset of AI focused on data-driven learning	Subset of ML focused on hierarchical feature learning
Goal	Develop intelligent agents to perform tasks like humans	Make data-driven predictions or decisions	Achieve high accuracy in complex tasks using multi-layer networks
Functionality	Thinks and acts like	Learns from data and	Automatically extracts

Aspect	Artificial Intelligence (AI)	Machine Learning (ML)	Deep Learning (DL)
	a human	improves over time	features and learns from unstructured data
Human Involvement	Often requires manual programming and logic-based rules	Requires data preparation and algorithm selection	Minimal feature engineering; high dependency on hardware and data
Data Requirements	Can work with structured knowledge or rules	Works well with structured data	Needs large amounts of labeled data for effective learning
Examples	Chatbots, autonomous robots, expert systems	Spam filters, recommendation systems, fraud detection	Image recognition, speech recognition, language translation

1.7. Practical Activities

1.7.1. Activity: "Recommend the Song" – Explore Recommendations

Objective: Demonstrate how ML powers recommendation systems like in YouTube or Spotify.

Instructions:

- Ask students to observe how their music apps suggest songs.
- Let each student list 3–5 recent songs they liked.
- Based on that, pair students and let them "recommend" songs to each other based on similar preferences.

Learning Outcome: Students understand how ML uses past behavior to predict future interests.

1.7.2. Activity: Create an ML Workflow Poster

Objective: Visualize the steps in building an ML model.

Instructions:

- Provide chart paper or ask for digital posters.
- Ask students to design the 9-step ML workflow:
- Problem Definition, Data Collection, Data Preprocessing, Data Analysis, Model Selection, Model Training, Model Evaluation, Model Deployment, Maintenance
- Use visual icons or drawings to make it engaging.

Learning Outcome: Reinforce the sequence and importance of each ML development stage.

1.7.3. Activity: AI vs. ML Sorting Game

Objective: Clarify the difference between AI and ML.

Instructions:

- Prepare 10 examples on slips of paper (e.g., Chatbot, Image Classifier, Self-driving Car, Rule-based System).
- Ask students to sort them into 3 categories:
  - AI
  - ML
  - Both
- Discuss the classification after sorting.

Summary

This session introduces Machine Learning (ML) as a subset of Artificial Intelligence (AI) that allows systems to learn from data and make predictions without explicit programming. Students learn the ML process through a fruit classification example (fresh vs. rotten). Key topics include workflow of ML (problem definition → data collection → preprocessing → model training → evaluation → deployment), benefits of ML (automation, efficiency, adaptability), and its wide applications in healthcare, banking, retail, transport, manufacturing, and more. Students also explore real-life applications like spam detection, product recommendations, voice assistants, and fraud detection. The session ends by differentiating AI, ML, and Deep Learning and includes practical classroom activities like building an ML workflow poster and AI vs. ML sorting games.

Check Your Progress

A. Multiple Choice Questions

1. Which of the following best describes the main goal of Machine Learning? (a) Store large amounts of data (b) Write fixed rules for all tasks (c) Learn from data and improve automatically (d) Build websites
2. Which step comes after data collection in the ML workflow? (a) Model training (b) Data preprocessing (c) Model deployment (d) Problem definition
3. Which of the following is not an application of Machine Learning? (a) Predicting stock prices (b) Writing novels creatively without data (c) Medical image analysis (d) Email spam filtering
4. Which of the following best explains "feature extraction" in ML? (a) Displaying results of predictions (b) Selecting useful input data characteristics (c) Deleting unwanted data (d) Saving model outputs
5. What type of machine learning is used when labels (output values) are not provided? (a) Supervised learning (b) Unsupervised learning (c) Reinforcement learning (d) Semi-supervised learning

B. Fill in the Blanks

1. A _____ is an algorithm that allows a machine to learn patterns from data.
2. In supervised learning, each training example is paired with an associated _____.
3. In the ML workflow, data _____ is done to remove noise and handle missing values.
4. The step where the model is tested on unseen data is called _____.
5. Machine Learning enables systems to make decisions without being _____ programmed.

C. State Whether True or False

1. Machine Learning models can update and improve with new data over time.
2. In Machine Learning, model training always comes before data preprocessing.
3. Classification is a type of supervised learning.
4. ML is not useful in applications that involve visual recognition.
5. In reinforcement learning, an agent learns by receiving rewards or penalties.

D. Short Answer Questions

1. Define feature extraction in the context of Machine Learning.
2. How is Machine Learning different from traditional programming?
3. What are the key benefits of using ML in the healthcare industry?
4. Name two algorithms used in classification problems.
5. Briefly describe the role of data preprocessing in ML.

Session 2. Types of Machine Learning

Think of how people learn: sometimes with guidance, sometimes by exploring, or sometimes through rewards. Machine Learning also works the same way. For instance, a student solving math sums with answers given (supervised), exploring a puzzle without hints (unsupervised), or improving in a video game by earning points (reinforcement). Deep learning goes further, like our brain using multiple layers of neurons to solve complex problems such as recognizing faces. A grocery store predicting which customers buy apples (supervised), Netflix grouping users by preferences (unsupervised), a dog trained with rewards (reinforcement), and Face ID on phones (deep learning).

Fig. 2.1

In this session, students learn about the main types of ML — Supervised (learning from labeled examples), Unsupervised (finding hidden patterns), Reinforcement Learning (learning from trial and error), and Deep Learning (using many layers like the human brain).

Types of Machine Learning

Machine Learning can be categorized into various types, including Supervised Learning, Unsupervised Learning, Reinforcement Learning, and Deep Learning. Figure 2.2 shows its hierarchy.

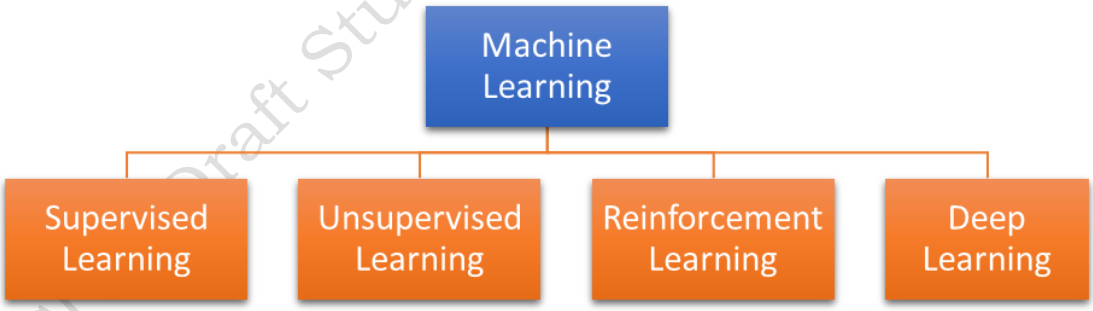


Fig. 2.2: Types of Machine Learning

2.1 Supervised Learning

Imagine teaching a friend to identify different types of flowers. You show them a rose and say, “This is a rose. It has red petals and thorns.” Then you show a sunflower and say, “This is a sunflower. It’s large, yellow, and has a big center.” After a few examples, your friend can name new flowers correctly. This is how supervised learning works in machine learning.

Supervised learning is a method where a computer learns to predict or classify things by studying examples. Each example has an input (like the flower’s features) and a correct

output (like the flower's name). The computer uses these examples to find patterns and make predictions on new data.

Example: Here's a tiny dataset for predicting exam results:

Study Hours	Result
2	Fail
5	Pass
1	Fail
6	Pass

The computer learns: "More study hours often lead to passing."

Key Components of Supervised Learning

Supervised learning involves several parts that are shown in Figure 2.3.

1. **Labeled Data:** This is the dataset with inputs and their correct outputs. It's like a guidebook for the computer.
2. **Model:** A mathematical system that learns patterns from the data.
3. **Training:** The process where the model studies the data to improve its predictions.
4. **Prediction:** The model uses what it learned to predict outputs for new inputs.

Example: the model is trained on labeled images of shapes like circles, triangles, and squares, and it learns to identify and classify new shapes based on those labels.

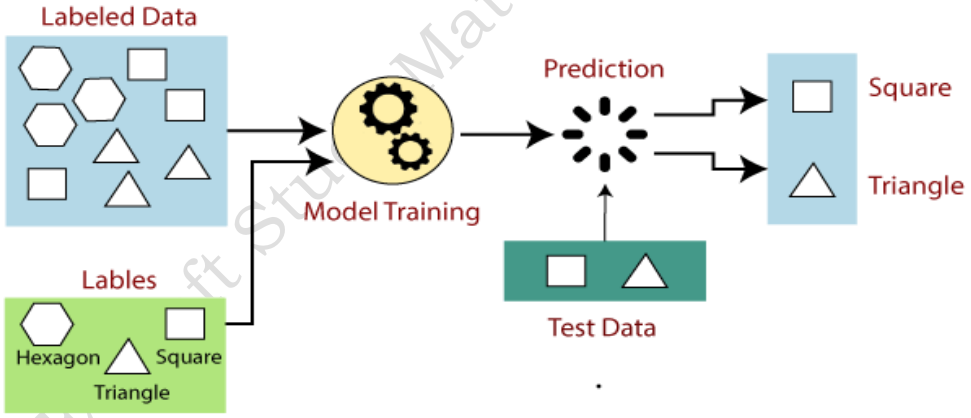


Fig. 2.3: Components of Supervised Learning

Real-Life Example

A grocery store uses customer purchase data as shown in Table 2.1, to predict if someone will buy apples. The labeled data includes what items customers bought and whether they included apples.

Table 2.1: Customer purchase dataset

Bought Bananas	Bought Oranges	Bought Apples
Yes	No	Yes
No	Yes	No
Yes	Yes	Yes

No	No	No

The model learns patterns, like “People who buy bananas are more likely to buy apples.”

Types of Supervised Learning

Supervised learning is split into two main types: classification and regression.

A. Classification

Classification involves sorting things into categories, like labelling something as “true” or “false.”

A classification problem arises when the outcome to be predicted belongs to a specific category or class, such as "red" or "blue," or "disease" versus "no disease." Classification falls under the domain of supervised learning and is commonly used to forecast discrete outcomes. Typical examples include determining whether a customer will cancel a service, whether an email is spam, or whether a medical image contains signs of a tumour. In this approach, classification algorithms learn to associate input features with a probability distribution across the possible categories, ultimately predicting the most likely class for a given input. The Figure 2.4 shows how classification works.

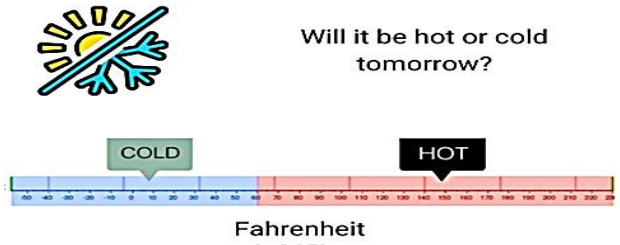


Fig. 2.4: Weather classification (Redraw)

Example: A school uses a system to classify students as “likely to join sports” or “not likely” based on their hobbies. For example, students who enjoy running might be more likely to join. Dataset for sports participation that is used to train the model is given in Table 2.2.

Table 2.2: Dataset for sports participation

Hobby	**Joins Sports**
Running	Yes
Reading	No
Swimming	Yes
Painting	No

The model learns: “Running and swimming hobbies suggest a student will join sports.”

B. Regression

A regression problem involves predicting a continuous, real-valued output—such as weight, temperature, or price. It is a form of supervised learning where the goal is to estimate numerical values based on input features. Common applications include forecasting housing prices, stock market trends, or estimating the likelihood of customer churn. Regression models work by learning a mathematical relationship between the

input variables and the continuous output, enabling accurate predictions for unseen data. The Figure 2.5 shows how classification works.

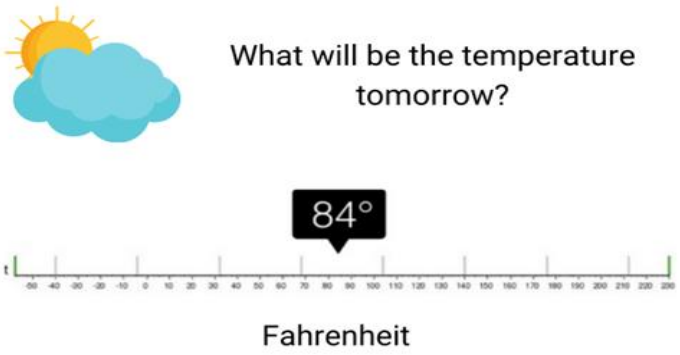


Fig. 2.5: Today's temperature (Redraw)

Example: A farmer predicts how many kilograms of tomatoes they'll harvest based on the amount of water used. More water might mean more tomatoes. Table 2.3 shows Dataset for tomato harvest, that is used to train the model.

Table 2.3: Dataset for tomato harvest

Water (Liters)	**Harvest (kg)**
10	5
15	7
20	9
12	6

The model learns: "More water generally increases the harvest."

2.2. Un-Supervised Learning

This is another category of machine learning that works with unlabeled data, meaning the model does not know the correct output during training. Instead of being told what to look for, it analyzes the data to uncover hidden patterns or structures on its own.

Example: Suppose a model is given a set of unlabeled images containing various shapes like circles, triangles, and squares as shown in Figure 2.4. Without knowing their names, the model groups them based on common features—such as the number of sides or overall shape—forming natural clusters.

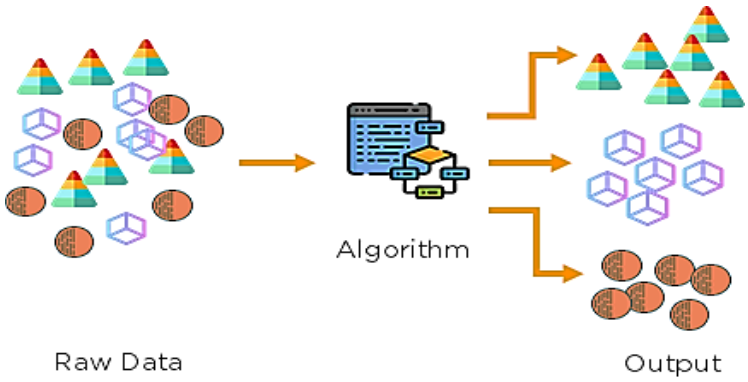


Fig. 2.6: Unsupervised learning (Redraw)

The main objective of unsupervised learning is to explore and identify inherent groupings or relationships within the data. This can include finding clusters, detecting outliers, or uncovering associations. Common unsupervised learning techniques include k-means clustering, hierarchical clustering, principal component analysis (PCA), and autoencoders.

How It Works:

11. **No Labels Provided**: The model receives images of geometric figures without any category names.
12. **Pattern Recognition**: It uses clustering algorithms (like K-means) to identify similarities, such as the number of edges or symmetry.
13. **Grouping**: The system forms clusters:
    - Shapes with 3 sides (triangles)
    - Shapes with 4 sides (squares or rectangles)
    - Round shapes (circles)
14. **Practical Use**: These groupings can be used in educational software to teach shape recognition or in design tools for categorizing objects.

More Examples of Unsupervised Learning

Anomaly Detection: Identifying unusual shapes or outliers in a batch, similar to spotting counterfeit coins in a collection.

Topic Modeling: Grouping text articles by themes without predefined categories—like sorting news by topic based on keyword patterns.

Recommendation Systems: Grouping users based on their behavior (like how they draw or choose shapes), and suggesting new learning activities accordingly.

Types of Unsupervised Learning

A. Clustering

Clustering is the process of grouping similar items together based on shared characteristics, without using any predefined labels.

Example: Customer Segmentation for a Supermarket

Imagine a supermarket that has data about its customers as shown in Table 2.4.

Table 2.4: Spending dataset of customers

Customer	Age	Monthly Spending (₹)
A	20	1500
B	35	6000
C	22	1800
D	40	5500
E	21	1700
F	45	5800

Using clustering (e.g., K-Means), the model may group:

Group 1: Young, low-spending customers (A, C, E)

Group 2: Older, high-spending customers (B, D, F)

Use Case:

The store can now send budget-friendly offers to Group 1 and premium deals to Group 2.

Common Algorithms:

K-Means

DBSCAN

Hierarchical Clustering

B. Dimensionality Reduction

Dimensionality reduction is a technique used to reduce the number of features (input columns) in a dataset while retaining its essential structure and patterns. It's useful for simplifying large datasets and improving data visualization.

Example: Student Performance Analysis

Imagine a school collects multiple performance indicators for each student as shown in Table 2.5.

Table 2.5: Students' performance indicators

Student	Attendance (%)	Homework Score	Project Score	Quiz Score
A	90	85	88	80
B	60	70	75	65
C	95	90	92	88

These features represent different aspects of student performance. Using Principal Component Analysis (PCA), this dataset can be reduced to 2 or 3 key dimensions that still capture most of the meaningful variation (e.g., "Engagement Level" and "Academic Strength").

Use Case:

This simplified dataset helps teachers visualize clusters of students (e.g., high performers vs. those needing support) without being overwhelmed by too many variables.

Common Techniques:

PCA (Principal Component Analysis) – to reduce dimensions while retaining variance.

t-SNE – for visualizing high-dimensional data in 2D or 3D.

Autoencoders – neural networks that learn compressed representations of data.

2.3. Reinforcement Learning

This type of learning is based on the idea of learning through trial and error. In reinforcement learning, a model (called an agent) interacts with an environment and learns by receiving rewards for correct actions and penalties for wrong ones.

Dog and Bell Example – Imagine training a dog to associate a bell with food, as shown in Figure 2.7. Initially, the dog only salivates when it sees food. Over time, if you ring a bell every time you offer food, the dog begins to associate the bell with the reward (food). Eventually, the dog starts salivating just upon hearing the bell—even if no food is immediately present. This shows that the dog has learned from repeated experience to expect a reward from a specific stimulus.

This process mimics reinforcement learning, where actions (like responding to the bell) are shaped by outcomes (like receiving food). The dog's behavior is reinforced through rewards, leading it to learn a pattern over time.

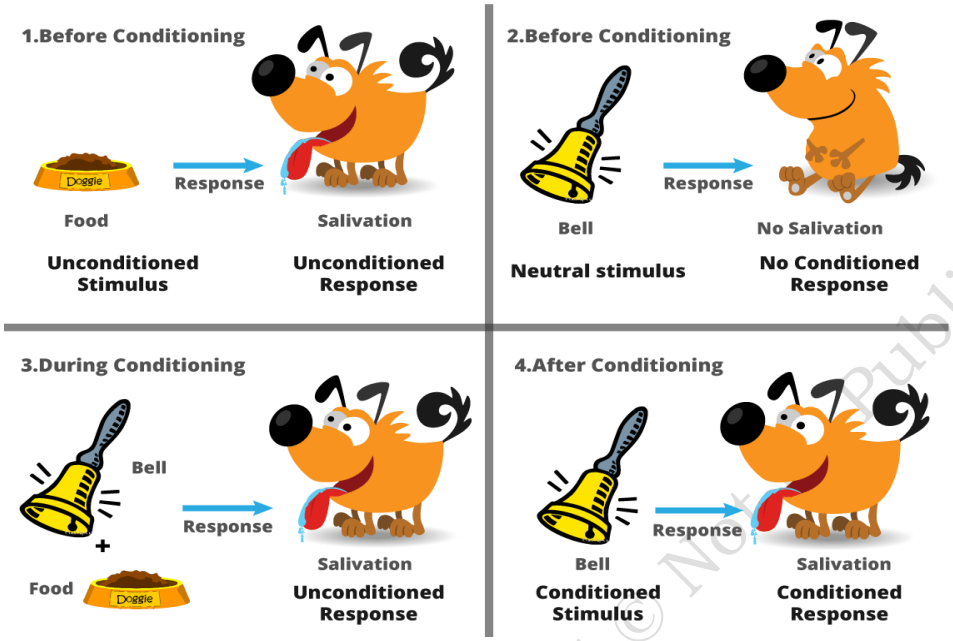


Fig. 2.7: Reinforcement learning of a dog

Reinforcement Learning (RL) is a type of Machine Learning where an agent learns to make decisions by interacting with its environment. The agent's goal is to maximize cumulative rewards over time. It does this by choosing actions that lead to positive outcomes and avoiding those that lead to penalties.

The learning process is dynamic — the agent continuously adjusts its strategy based on the feedback it receives from the environment.

How It Works:

- Agent: The dog
- Environment: The space in which the dog hears the bell and gets food
- Action: Salivating or responding when it hears the bell
- Reward: Receiving food
- Penalty: No food when the wrong action is taken (e.g., ignoring the bell)

Over time, the dog learns to associate the bell with food and begins to respond more accurately — just like an RL agent refining its strategy.

Reinforcement Learning in the Real World

Just as a dog learns behaviour through feedback, RL is used to train machines in scenarios where continuous decision-making is essential. Some applications include:

1. ***Self-driving Cars***: Learning to drive safely by interacting with traffic environments
2. ***Game AI***: Developing strategies to beat human players in games like Chess or Go
3. ***Robotic Control:*** Teaching robots to perform tasks like picking objects or walking without falling

4. ***Finance:*** Building trading bots that learn to make profitable investment decisions

Popular Reinforcement Learning Algorithms

a) Q-Learning
b) Deep Q-Networks (DQN)
c) Policy Gradient Methods
d) Actor-Critic Models

2.4. Deep Learning (DL)

Deep Learning is a specialized area of Machine Learning that uses networks with many layers, known as deep neural networks, to solve complex problems. These models try to learn in a way similar to how the human brain learns from repeated practice.

2.4.1. Working Process of Deep Learning

In traditional machine learning, a human expert usually has to decide what features of the data are important. For example, in identifying a cat, a human might highlight features like whiskers, sharp ears, or tail shape.

In contrast, deep learning models learn these features automatically. The first layers may detect simple patterns such as edges and colors, while deeper layers recognize higher-level details like shapes, objects, or even complete faces, as shown in Figure 2.8: Structure of a Deep Neural Network.

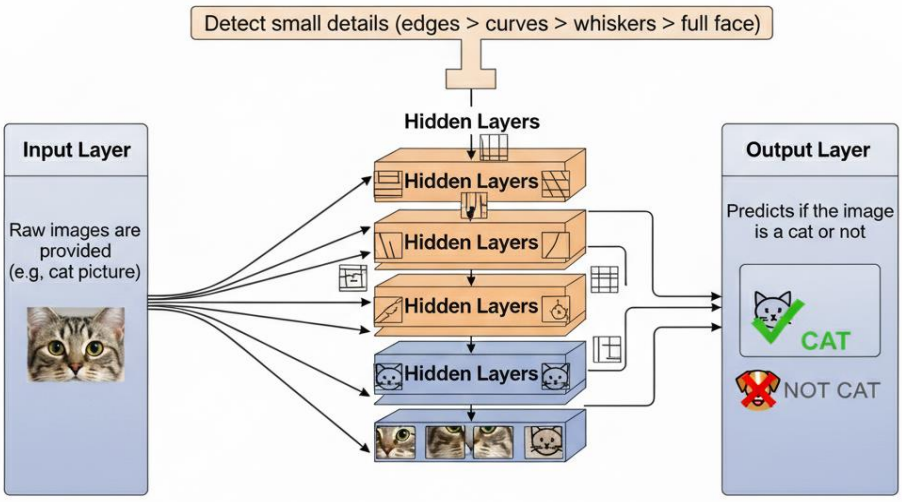


Fig. 2.8: Structure of a Deep Neural Network

The term **"deep"** refers to the presence of multiple layers in the neural network. While shallow networks (with only one or two hidden layers) can solve simple problems, deep networks with many layers can understand more complex patterns.

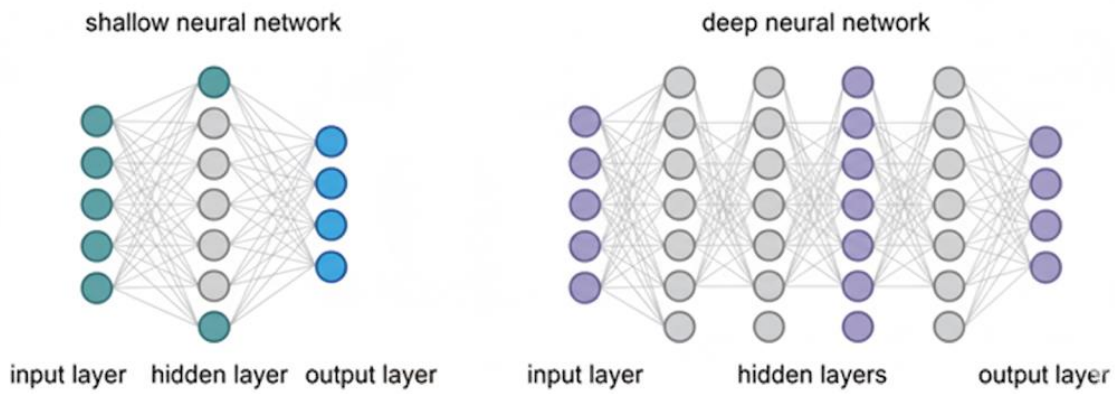


Fig. 2.9: Shallow vs. Deep Neural Network (Redraw this Figure)

2.4.2. Machine Learning vs. Deep Learning

Although both belong to the same family, deep learning is designed for more data-intensive and complex tasks. See the differences in various aspects in Table 2.6.

Table 2.6: Difference Between Machine Learning and Deep Learning

Aspect	Machine Learning	Deep Learning
Feature Selection	Manual, done by experts	Automatic, learned by the network
Dataset Requirement	Can work on smaller datasets	Requires large datasets
Computing Power	Runs on normal computers	Needs advanced hardware (GPUs/TPUs)
Applications	Spam filtering, product recommendations	Face recognition, driverless vehicles

2.4.3 Real-World Applications of Deep Learning

A. Face Recognition: Many smartphones unlock using Face ID, which compares the user's face in real time with stored data using deep learning (Figure 2.10).

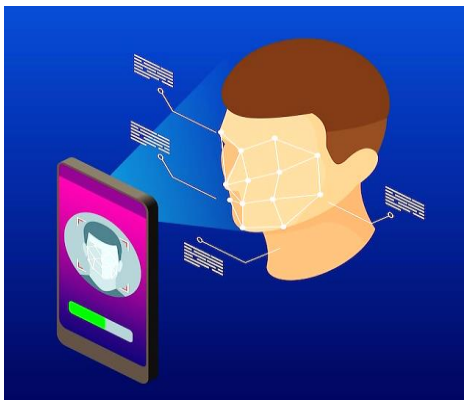


Fig. 2.10: Face recognition process using deep learning

B. Self-Driving Cars: Autonomous vehicles rely on deep learning to detect traffic signs, people, and obstacles in real time. (Figure 2.11).

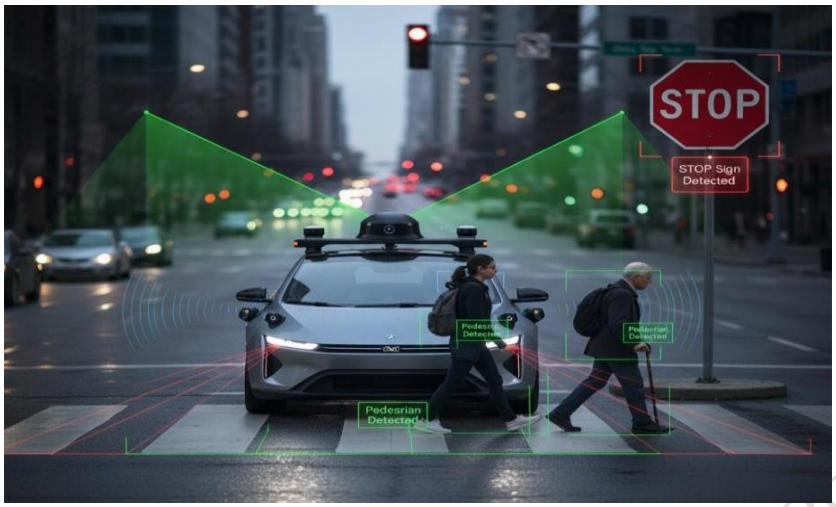


Fig. 2.11: Car detecting road signals and pedestrians using sensors

C. Healthcare: Doctors and researchers apply deep learning to study medical images such as X-rays or MRI scans for faster and more accurate diagnoses. The diseases identified with deep learning are given in Table 2.7.

Table 2.7: Diseases identified with deep learning

Medical Domain	Disease/Condition	Deep Learning Application
Respiratory System	Pneumonia	Detection from chest X-rays and CT scans
Dermatology	Skin Cancer (Melanoma)	Image classification of skin lesions
Neurology	Brain Tumors	MRI scan segmentation and tumor localization
Ophthalmology	Diabetic Retinopathy	Fundus image analysis for early detection
Cardiology	Heart Disease (Arrhythmia)	ECG signal classification and cardiac MRI analysis
Pathology	Breast Cancer	Histopathology slide analysis for malignancy detection

D. Language Translation: Applications like Google Translate use deep learning to convert text or speech instantly from one language to another.

2.4.5. Important Features of Deep Learning

a) **Learns automatically from raw data** without the need for hand-crafted rules.
b) **Improves performance** as the amount of data increases.
c) **Versatile** in working with text, images, audio, or video.
d) **High accuracy** when trained with sufficient data and computing power.

2.4.6. Challenges of Deep Learning

Deep Learning is powerful, but still it has certain limitations as stated below:

a) Needs **large datasets** for good performance
b) Requires **high computing power** (special processors like GPUs)
c) Training can be **time-consuming**

d) Difficult to explain why the model made a particular decision (**black box issue**)

Table 2.3 Placeholder: Advantages and Limitations of Deep Learning

Advantages	**Limitations**
Learns directly from data	Needs large labeled datasets
Delivers high accuracy	Demands powerful hardware
Works across many domains	Difficult to interpret model results

2.5. Practical Activities

Activity 2.1. Identify and Predict

Topic: Supervised Learning – Classification

Objective: Understand how classification models work using labeled data.

Material: Chart paper, pictures of animals or fruits, markers

Fig. 2.12: Identify and Predict Activity

Procedure

Step 1. Collect or draw pictures of 10 different animals or fruits (e.g., Apple, Banana, Lion, Tiger).

Step 2. Label each image as "Fruit" or "Animal".

Step 3. Mix the images and ask your friends to classify them based on past examples.

Step 4. Discuss how your friend (like a model) learned to classify based on labeled training data.

Step 5. Record their classification accuracy.

Activity 2.2. Grouping

Topic: Unsupervised Learning – Clustering

Objective: Learn how to group similar items without labels.

Material: A mix of classroom objects (erasers, pencils, markers, clips, etc.)

Fig. 2.13: Group Activity

Procedure

Step 1. Collect 15 classroom items with no labels.

Step 2. Observe the features: colour, size, shape, use.

Step 3. Group them into 2–3 clusters based on similarities (e.g., writing tools, stationery, sharp items).

Step 4. Explain the logic behind your grouping.

Step 5. Compare your grouping with a friend's.

Learning Outcome: Understand how unsupervised models find patterns or clusters in data.

Activity 2.3. Predicting with Numbers

Topic: Supervised Learning – Regression

Objective: Learn how increasing one factor affects the result.

Material: Record of students' study time and their test marks (fictional or real anonymized data).

Sample Dataset:

Study Time (hrs)	Marks (%)
1	40
2	50
3	60
4	70

Fig. 2.14: Predicting with Numbers Activity

Procedure

Step 1. Plot this data on graph paper or use a digital spreadsheet.

Step 2. Draw a line that best fits the data (trendline).

Step 3. Predict marks if a student studies for 5 hours.

Step 4. Observe that "more study hours → better performance".

Activity 2.4. Behaviour Training Game

Topic: Reinforcement Learning

Objective: Experience learning from trial and error.

Material: A simple maze drawn on paper or created with books/blocks, a toy robot or paper pointer

Fig. 2.15: Behaviour Training Game Activity

Procedure

Step 1. Create a maze with a clear goal (e.g., reach a flag).

Step 2. Let a classmate (the "agent") try different paths.

Step 3. Give +1 for each correct turn, -1 for wrong.

Step 4. After multiple attempts, the student learns the correct path.

Step 5. Discuss how rewards and penalties shaped the final behavior.

Activity 2.5. Dimension Reduction using Charts

Topic: Dimensionality Reduction

Objective: Reduce data while preserving meaning.

Material: Data on student activities (e.g., sports, dancing, music, drawing), chart paper

Sample Data:

Student	Sports	Music	Dancing	Drawing
A	9	3	5	2
B	2	8	4	7

C	8	2	3	1

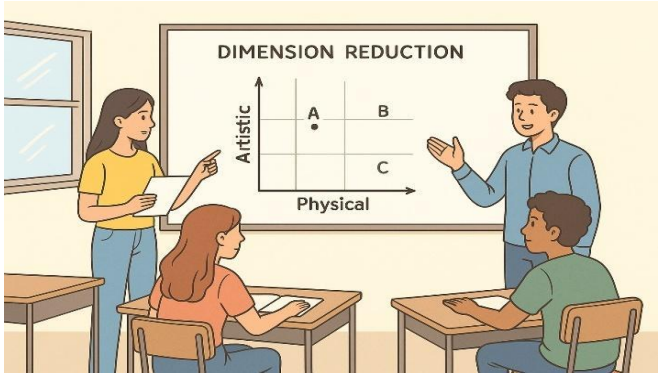


Fig. 2.16: Dimension Reduction using Charts Activity

Procedure

Step 1. Look for which activities seem related (e.g., sports and music are not related, music and drawing might be).

Step 2. Reduce the table to 2 main interests per student (e.g., combine music and drawing as "Artistic", sports and dancing as "Physical").

Steps 3. Plot a chart with these 2 new dimensions.

Summary

- Supervised Learning – learning from labeled data to perform classification (predicting discrete categories) and regression (predicting numerical values).
- Unsupervised Learning – working with unlabeled data to find hidden patterns, using techniques like clustering and dimensionality reduction.
- Reinforcement Learning – learning through trial and error, guided by rewards and penalties (like training a dog).
- Deep Learning – a branch of ML that uses deep neural networks to automatically extract features and solve complex problems like image recognition and natural language processing.
- Students learn with relatable examples (e.g., predicting exam results, clustering customers, reinforcement learning games). Hands-on activities include grouping items, predicting study outcomes, and simulating behavior training.

Check Your Progress

A. Multiple Choice Questions

1. Which of the following best defines supervised learning? (a) Learning without using labeled data (b) Learning by receiving penalties only (c) Learning from labeled data to make predictions (d) Learning by trial and error

2. What is the primary goal of unsupervised learning? (a) Predict future outputs (b) Group similar data points (c) Train models on labeled datasets (d) Control robots directly
3. Which type of ML is best suited for recommending content based on user behavior patterns? (a) Supervised Learning (b) Unsupervised Learning (c) Reinforcement Learning (d) Deep Learning
4. In reinforcement learning, what guides the agent's learning process? (a) Number of training epochs (b) Presence of labeled data (c) Rewards and penalties (d) Manual correction
5. Which of the following problems is most suitable for regression? (a) Classifying spam emails (b) Predicting house prices (c) Grouping customers (d) Identifying handwritten digits

B. Fill in the Blanks

1. In _____ learning, the model receives both input and the correct output to learn patterns.
2. _____ learning helps in grouping data without any prior labels or categories.
3. In reinforcement learning, an agent receives a _____ for taking a beneficial action.
4. _____ and regression are two types of problems in supervised learning.
5. K-means is a commonly used algorithm in _____ learning.

C. State Whether True or False

1. Supervised learning requires labeled data for training.
2. Dimensionality reduction increases the number of features in a dataset.
3. Reinforcement learning relies on feedback to improve decisions over time.
4. Unsupervised learning uses pre-defined output labels during training.
5. Classification problems involve predicting discrete categories.

D. Short Answer Questions

1. What are the two main types of problems solved using supervised learning?
2. Explain the role of the agent in reinforcement learning.
3. How does clustering work in unsupervised learning?
4. Give a real-life example of a classification problem.
5. What is the key difference between supervised and unsupervised learning?

Session 3. Data in Machine Learning

3.0 Introduction

Data is the fuel for ML. Just as a chef needs quality ingredients to cook a tasty dish, ML needs clean and organized data to give accurate results. For example, an e-commerce site like Amazon uses customer shopping history (features) and actual purchases (labels) to recommend new products. If the data is messy or wrong, the recommendations will also fail. YouTube recommends videos by analysing your watch history (features) and feedback (likes/dislikes = labels). Clean, correct data ensures better recommendations.

Fig. 3.1: Importance of Data for Machine Learning

This session focuses on the role of data in ML. Students learn about features (inputs), labels (outputs), training and testing data, and the importance of clean and sufficient data.

3.1 Importance of Data in Machine Learning

Imagine trying to learn something new—like how to play cricket or bake a cake—without seeing any examples or practicing. It would be nearly impossible. Just like humans need information and practice to learn, machines also need examples to learn. This is where data comes into play in Machine Learning (ML).

Data is the foundation of every Machine Learning model. Without data, a machine cannot learn patterns, make predictions, or improve performance. In this chapter, we will explore what data means in ML, what types of data are used, how it's divided for learning, and why it must be clean and sufficient. You will also get hands-on experience with a small project and practical exercises.

Think of apps you use every day—YouTube, Instagram, Amazon. They recommend videos or products based on your data. They analyze what you watched, liked, or bought before. This is Machine Learning in action, powered by your data.

3.2 Features and Labels

In Machine Learning (ML), two key terms appear repeatedly: features and labels. These form the foundation of every ML problem. Without clearly understanding them, it is difficult to make sense of how machines learn.

3.2.1. Features: The Inputs for Learning

Features are the input variables that we provide to a machine learning model. They are measurable, observable properties of the data that help the system make predictions or classifications.

Think of features as the "clues" that help the machine arrive at an answer. The quality and relevance of features strongly influence how accurate the model will be.

Imagine you are a teacher trying to guess which students are likely to do well in the final exam. You may look at factors such as:

1. How many hours the student studied daily
2. Attendance percentage in class
3. Whether homework was regularly completed
4. Participation in group discussions

Observe the Figure 3.1, the pieces of information act as features, because they describe the situation and help in predicting an outcome.

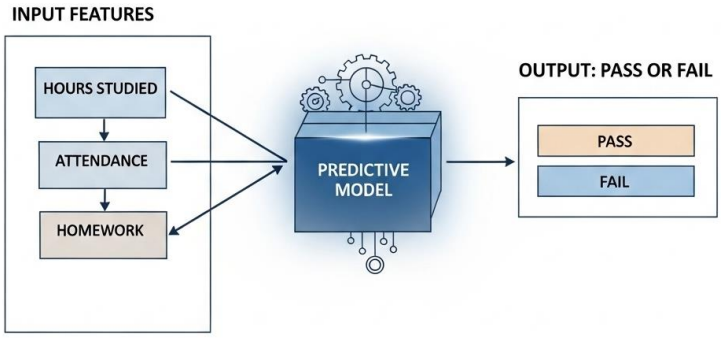


Fig. 3.1: Input Features to predict Pass or Fail

3.2.2. Labels: The Outcomes

A label is the final output or result that the machine is trained to predict. It answers the question that we are asking the machine.

Continuing the earlier example of predicting exam results:

- The label is simply whether the student Passed or Failed.

Thus, while features represent the input conditions, the label represents the expected answer.

Real-Life Analogy:

Think of a doctor predicting whether a patient has a fever, as shown in Figure 3.2.

1. Features: Body temperature, sore throat, headache, fatigue.
2. Label: "Fever" (Yes/No).

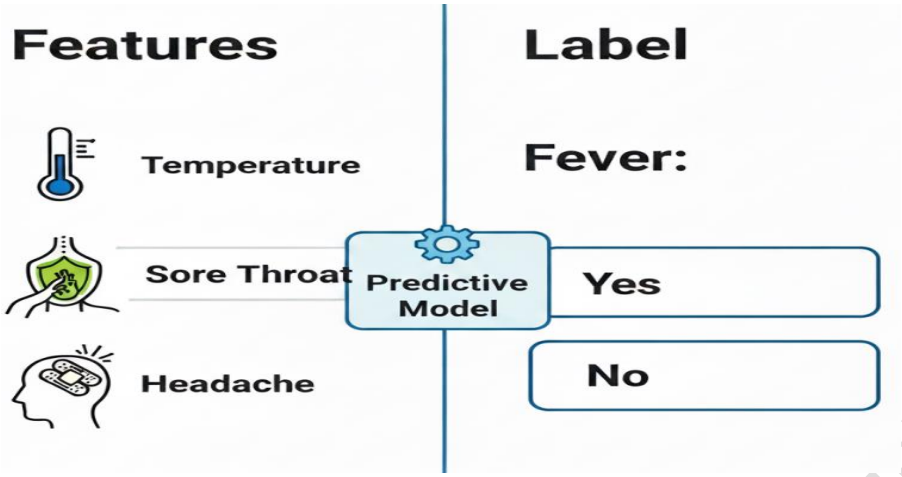


Fig. 3.2: Features and Label

Example Dataset

To understand this better, let us look at a tiny dataset in Table 3.1, about students' performance:

Table 3.1: Example Dataset

Hours Studied	Attendance (%)	Homework Done	Result (Label)
2	60	No	Fail
5	90	Yes	Pass
1	50	No	Fail
6	95	Yes	Pass

1. Here, the first three columns (Hours Studied, Attendance, Homework Done) are the features.
2. The last column (Result) is the label.

3.2.3. Importance of Features and Labels

1. Features give context. They tell the model what information to use for making predictions.
2. Labels provide direction. They tell the model what the correct answer should be.
3. Without features, the model has no data to work on. Without labels, the model cannot evaluate its predictions during training.

Example:

If we only had student study hours and attendance, but no information about whether they passed or failed, the model would not know what it is supposed to predict.

3.2.4. Types of Features

Not all features are the same. In ML, features may be classified into different types:

a) Numerical Features: Quantities or measurements (e.g., hours studied = 5).
b) Categorical Features: Non-numerical categories (e.g., homework = Yes/No).
c) Ordinal Features: Categories with an order (e.g., grades = A, B, C).

3.2.5. Case Study Example: Predicting Sports Team Selection

Suppose a school wants to predict whether a student will be selected for the basketball team.

Features:

1. Height (cm)
2. Practice hours per week
3. Number of games played last year

Label: Selected (Yes/No)

By providing this dataset to a machine learning model, the school can predict which students are most likely to be selected based on past patterns.

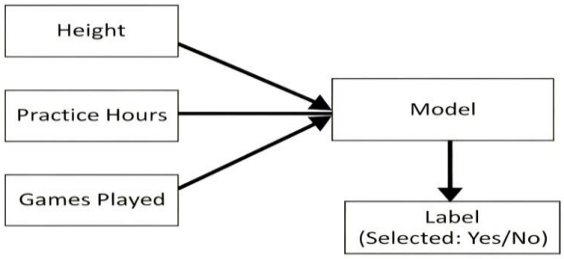


Fig. 3.3: Flowchart machine learning model

3.3. Training and Testing Data

In order to make predictions, a machine must first learn from data. This is called training. But just like students need to take exams to check their understanding, machines also need to be tested.

Training data is the data used to teach the machine. It includes both features and labels. The machine studies this data to learn patterns.

Testing data is new data that the machine has never seen before. It is used to evaluate how well the machine has learned from the training data. It also includes features, but the machine has to predict the labels.

If we only test the machine on the data it was trained on, we won't know how well it performs on new, unseen data. That's why we divide the data into two parts:

- **Training Set** (usually 70–80% of data)
- **Testing Set** (usually 20–30% of data)

Example: Look at Table 3.2, the machine will learn from the first three rows and try to predict the result of the last row.

Table 3.2: Training and testing data

Hours Studied	Attendance (%)	Homework Done	Result (Label)	Type
2	60	No	Fail	Training
5	90	Yes	Pass	Training
1	50	No	Fail	Training
6	95	Yes	Pass	Testing

3.4. Importance of Clean and Sufficient Data

Not all data is useful. Imagine trying to solve a math problem with incorrect or missing values. Similarly, if we give bad data to a machine, it will learn the wrong things.

3.4.1 Clean Data

Clean data means data that is correct, consistent, and well-organized. Dirty or messy data can lead to poor model performance.

Common Problems in Unclean Data:

1. **Missing values** (e.g., no attendance recorded)
2. **Incorrect entries** (e.g., "Yess" instead of "Yes")
3. **Duplicates** (same row repeated)
4. **Inconsistent formats** (e.g., dates in different styles)

3.4.2 Methods of Data Cleaning:

1. Fill in missing values
2. Remove duplicates
3. Standardize formats
4. Correct typos

3.4.3 Sufficient Data

Having too little data is like studying only one question before an exam. The machine will not learn enough patterns and will perform poorly. It is better to have a small, clean dataset than a large, messy one.

3.5. Practical Activity

Let's put theory into action. We will do some small practical activities that can be done using a low-end PC and common software like Excel or Google Sheets.

3.5.1. Activity: Exploring a CSV File in Excel

Objective:

To understand how data looks in tabular form and identify features and labels.

Steps:

1. Open Microsoft Excel or Google Sheets.
2. Create the following table or import from a small CSV file.

See the Sample CSV Data in the Table 3.3.

Table 5.3.3: Students dataset

Name	Study Hours	Attendance (%)	Homework Done	Passed
Amit	2	60	No	No
Sara	5	90	Yes	Yes
Raj	1	50	No	No
Anjali	6	95	Yes	Yes

1. Label the columns: Features = Study Hours, Attendance, Homework Done. Label = Passed.

2. Highlight the label column in a different color.

3.5.2. Activity: Manual Labeling of Data

Objective:

To understand how humans, help machines learn by labelling data.

Task:

Label this simple data as given in Table 3.4.

Table 3.4: Weather dataset

Weather	Mood
Sunny	?
Rainy	?
Cloudy	?
Sunny	?

Your labels might be:

Sunny → Happy

Rainy → Sad

Cloudy → Neutral

Manually fill in the "Mood" column of Table 5.3.4, based on your experience. This is what a human would do before giving the data to the machine for training.

3.5.3. Activity: Mini Project – Mood Prediction Based on Weather

Objective:

To create a simple ML dataset and visualize it.

Steps:

1. Collect your own data for 5 days as in Table 3.5.

Table 3.5: Weather dataset

Date	Weather	Mood
01/07/2025	Sunny	Happy
02/07/2025	Rainy	Sad
03/07/2025	Cloudy	Neutral
04/07/2025	Sunny	Happy
05/07/2025	Rainy	Sad

2. Open this in Excel or Google Sheets.
3. Create a bar chart:
   - X-axis: Weather
   - Y-axis: Mood count
4. Observe the pattern: Is your mood more cheerful on sunny days?

This helps understand how real-life patterns can be turned into datasets and learned by machines.

Summary

This session emphasizes the role of data as the foundation of ML. Features and Labels: Features are input variables (study hours, attendance), and labels are target outputs (Pass/Fail).

Training vs. Testing Data: Training data teaches the model; testing data checks how well it works on unseen inputs. Clean and Sufficient Data: Students learn the importance of accuracy, completeness, and consistency. Problems in unclean data include missing values, duplicates, and types.

Real-life connections are made with apps like YouTube and Amazon, which use user data to make recommendations. Activities include working with CSV files in Excel/Google Sheets, manual labelling, and a mini weather–mood prediction project.

Check Your Progress

A. Multiple Choice Questions (MCQs)

1. What is a feature in machine learning? (a) The final prediction made by the model (b) A tool to clean data (c) An input variable used to make a prediction (d) An error in the system
2. What is a label in a dataset? (a) A type of chart (b) The result the machine tries to predict (c) A column heading (d) A data type
3. Why do we split data into training and testing sets? (a) To save computer memory (b) To make learning fun (c) To evaluate how well the model performs on new data (d) To reduce file size
4. What happens if we train a model using unclean data? (a) The model becomes more intelligent (b) The predictions become more accurate (c) The model may learn incorrect patterns (d) The data automatically corrects itself
5. What is the ideal property of a dataset used in ML? (a) Large and random (b) Small and messy (c) Clean and sufficient (d) Only includes images

B. Fill in the Blanks

1. In machine learning, _____ are the input values used to make predictions.
2. The _____ is the output or target value that the machine tries to learn.
3. Data used to train a machine learning model is called _____ data.
4. Clean data is _____, consistent, and complete.
5. A small, _____ dataset is better than a large, messy one.

C. State Whether True or False

1. Features are used as input for training a model.
2. Testing data is the same as training data.
3. Labels are not necessary in supervised learning.
4. Data should be cleaned before training a machine learning model.

5. You can use Excel or Google Sheets to view and modify CSV data.

D. Short Answer Questions

1. What are features and labels in a dataset?
2. Why is it important to split data into training and testing sets?
3. What problems can occur if the data is not clean?
4. Give one example of a small dataset you can collect for a mini-project.
5. How can you manually label a dataset?

Session 4. Ethics and Responsibility in AI/ML

Using ML responsibly is very important. Imagine if a school grading system only trained on data from a few students — it may unfairly judge others. Similarly, AI tools can become biased if data is unbalanced. Ethical use of AI ensures fairness, transparency, and privacy. For example, face recognition apps should work equally well for people of all skin tones, not just some.

An AI hiring tool that wrongly prefers men over women due to biased past data. Responsible AI must ensure fairness and equal treatment for all.

Fig. 4.1: Responsible AI

This session highlights the importance of fairness, transparency, privacy, and unbiased data in AI/ML. It shows how wrong or unfair data can lead to unfair predictions. It helps you understand how to use AI/ML responsibly, avoid mistakes and bias, and respect privacy and fairness.

Ethics in Artificial Intelligence (AI) and Machine Learning (ML)

Artificial Intelligence (AI) and Machine Learning (ML) are powerful tools that are changing how we live, work, and make decisions. From recommending videos online to helping doctors diagnose diseases, these technologies are being used in many areas of life. However, with great power comes great responsibility. Just like humans, machines must also behave in a way that is fair, honest, and respectful of people's rights.

Example: Socioeconomic Bias in Viva Performance Prediction by ML, see Table 4.1.

Table 4.1: Student dataset

Student ID	**Family Income Level**	**Actual Viva Performance**	**ML Model Prediction**	**Remarks**
S1	High (Rich)	Average	Pass	✗ Biased: Passed despite average
S2	High (Rich)	Good	Pass	☑ Fair prediction
S3	High (Rich)	Poor	Pass	✗ Biased: Wrong pass
S4	Low (Middle-Class)	Good	Fail	✗ Unfair: Deserved to pass
S5	Low (Middle-Class)	Excellent	Fail	✗ Strong bias: Misjudged
S6	Low (Middle-Class)	Average	Fail	☑ Fair prediction

4.1 Bias in Data and Its Impact

Bias in ML means the model makes unfair or incorrect predictions because of problems in the data it was trained on. Bias can happen when:

1. The data mostly represent one group and ignore others. It can cause unfair treatment of people.
2. Some examples are missing or wrong. It might lead to wrong decisions that hurt someone's life.
3. The labels given to data are incorrect or unfair. It reduces the trust people have in AI/ML systems.

Examples of Bias

a) Facial recognition may work well for one skin tone but not for others if it was trained mostly on one type of image.

b) Loan approval systems might reject people from certain areas unfairly if past data includes such patterns.

4.2 Importance of Clean and Balanced Data

For AI/ML to work properly, the data must be:

1. Accurate
2. Complete
3. Representative of all groups
4. Free from errors or duplication

If the data is not clean, even the best model will make mistakes, as seen in Table 4.2.

Table 4.2: Clean Data vs. Dirty Data

Student Name	Hours Studied	Passed Exam	Remarks
Priya	4	Yes	✅ Clean entry
Rohan	6	Yes	✅ Clean entry
Anjali	2	No	✅ Clean entry
Ravi	?	No	❌ Missing value
Priya	4	Yes	❌ Duplicate entry
Simran	3	Yes	❌ Possibly mislabelled (low study hours)

Steps to Ensure Clean Data

a) Remove duplicates
b) Fix missing values
c) Check for errors in data entry
d) Ensure data covers all categories fairly

Imbalanced Data Example

Suppose you are training a model to identify if someone likes outdoor activities. If your dataset includes:

- 90% sports lovers
- 10% readers

Then your model might learn to think everyone loves sports. That's called data imbalance.

4.3 Fairness, Transparency, and Explain ability

4.3.1 Fairness

Fairness means that the AI/ML system treats everyone equally, regardless of gender, race, age, or background.

Unfair ML in Real Life

- A job application system giving better chances to men over women.
- A school admission tool preferring urban students over rural ones.

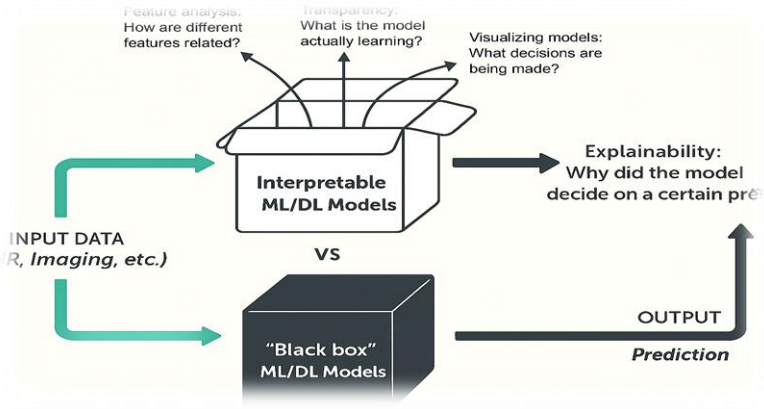
To avoid such problems, fairness must be checked during training.

4.3.2 Transparency

Transparency means being clear about:

a) How a model works
b) What kind of data it uses
c) What decisions it makes and why

When an AI/ML system is not transparent, people cannot trust it.



4.3.3 Explain ability

Explain ability is the ability to understand why an AI system made a certain decision.

Example:

If an ML model predicts that a student won't pass an exam, we must know:

1. Was it because of attendance?
2. Low quiz scores?
3. Or something else?

Without explain ability, mistakes can't be corrected, and people won't feel safe using such systems.

4.4 Privacy and Data Security

4.4.1 Privacy in AI/ML

Privacy means keeping a person's data safe and using it only with their permission. AI/ML models are often trained using personal data like: Age, Address, Health records, and Online activity. This data must be protected to avoid harm.

Importance of Privacy

1. Prevents identity theft
2. Avoids unwanted marketing or surveillance
3. Respects individual rights

4.4.2 Data Security

Data security is about keeping personal information safe from hackers, leaks, or misuse.

Good security ensures:

1. Data is stored safely
2. Access is limited to authorized users
3. Information is encrypted (locked) during storage and sharing

4.5 Real-Life Case: Biased Hiring Tool

A large tech company built an AI tool to shortlist resumes for jobs. But the system kept selecting more men than women. Why?

- The training data came from past hiring records.
- The company had mostly hired men in the past.
- The model learned to copy those decisions, assuming men are better hires.

This case shows how past bias can repeat in AI, making it unfair.

Solution of Bias

- Use clean and balanced data.
- Check the model results for fairness.
- Remove any sensitive personal information from training data.

4.6 The AI Responsibility Checklist:

1. Ask Questions About Data

- Is it balanced?
- Does it include all groups?
- Are there any errors?

2. Include People from All Backgrounds

A team of designers, coders, and testers should be diverse. This ensures that different views are considered during model building.

3. Test Results Fairly

Check how well the model performs for all groups:

- Does it work for girls and boys equally?
- Is it fair to all communities?

4. Be Honest About Limitations

No model is perfect. Always explain where the model might go wrong.

4.7 Practical Activities

4.7.1 Activity: Accuracy with Clean vs. Dirty Data

Objective: See how clean data improves ML predictions.

Dataset: as shown in Table 4.3.

Table 4.3: Student study-result dataset

Hours Studied	Pass/Fail
2	Fail
5	Pass
3	?
6	Pass
?	Fail
1	Fail
5	Pass

Instructions:

- Open the file in Excel or Google Sheets.
- Fill in missing data and remove the question marks.
- Replace "?" with actual values (like 4 for hours, and "Pass" if it makes sense).
- Observe how cleaning helps in building better models.

4.7.2 Activity: Watch and Reflect on AI Bias

Objective: Understand how bias affects real people.

Instructions:

- Watch a short 5-minute video (suggested link or documentary name can be added by school/teacher).
- After watching, answer:
  - What was the bias shown?
  - Who was affected?
  - How can this be prevented?

4.7.3 Activity: Demonstrate the Importance of Clean Data

Create a small dataset in Excel or Google Sheets using Table 4.4.

Table 5.4.4: Student basic details dataset

Name	Age	Gender	Passed (Yes/No)
Ravi	16	Male	Yes
Anu	15	Female	No
Ramu		Male	Yes
Anita	17	Female	Yes
Ravi	16	Male	Yes

Step 1. Observe the dataset and identify errors (missing age, incorrect spelling "Yes", duplicate rows).

Step 2. Clean the data by fixing or removing mistakes.

Step 3. Discuss how cleaning improves data quality and helps get accurate Check in your Progress

Summary

This session covers the importance of responsible AI/ML practices. Key issues:

- Bias – occurs when training data is unbalanced or unfair, leading to incorrect predictions (e.g., biased hiring or loan approvals).
- Fairness – ensuring AI treats all groups equally.
- Transparency & Explain ability – making models understandable and trustworthy by showing how decisions are made.
- Privacy & Data Security – keeping user data safe, accurate, and protected from misuse.

  Real-life cases (biased hiring tool, facial recognition issues) are discussed. Students are introduced to the AI Responsibility Checklist (clean data, diverse teams, fair testing, honesty about limitations). Activities involve comparing clean vs. dirty data, watching bias-related videos, and cleaning small datasets to observe improvements.

Check Your Progress

A. Multiple Choice Questions

1. What is one major cause of bias in ML systems? (a) Too many examples (b) Incorrect coding (c) Unbalanced training data (d) Fast internet
2. Why is transparency important in ML? (a) To keep models secret (b) To confuse users (c) To build trust by explaining decisions (d) To reduce speed
3. What is privacy in AI? (a) Hiding code from users (b) Keeping personal data safe (c) Giving rewards to users (d) Speeding up the process
4. Which of the following helps improve fairness? (a) Using old data without checking (b) Including all user groups (c) Ignoring errors (d) Only testing once
5. What is a key part of explain ability? (a) Making random predictions (b) Showing how decisions were made (c) Hiding model results (d) Encrypting passwords

B. Fill in the Blanks

1. Bias in machine learning often comes from _____ data.
2. _____ means understanding how an AI model made its decision.
3. Keeping user data safe is known as _____.
4. _____ is about treating all individuals equally in ML systems.
5. In a responsible ML project, we should use data that is _____ and representative.

C. State Whether True or False

1. An ML model should always be tested for fairness.
2. Transparency in AI means hiding the way it works.
3. Explain ability helps users trust AI systems.
4. Dirty or incomplete data improves ML results.
5. AI should be trained only on data from one group of people.

D. Short Answer Questions

1. What is bias in machine learning? Provide an example.
2. Why is clean data important for building ML models?
3. How does fairness improve trust in AI systems?
4. Define transparency and explain ability in your own words.
5. Describe one way to protect data privacy in ML systems.

Session 5. Hands-On ML Without Code (Using Tools)

5.0 Introduction

ML is not just theory — you can try it practically. For example, with Teachable Machine, you can train your computer to recognize hand gestures like “peace sign” or “thumbs up” simply by showing it a few examples. No programming is needed, but the experience helps you understand how ML works step by step.

Using Teachable Machine, you can train your webcam to recognize a “thumbs up” or “thumbs down” gesture in just a few minutes.

This session teaches students how to try ML practically using easy, no-code tools like Teachable Machine and Google Colab. They can train and test models with images, sounds, or numbers without programming.

5.1 Training and Testing in ML

Machine Learning (ML) is not just for expert programmers. With the help of simple online tools, anyone can build and test ML models without writing code. In this chapter, you will learn how to train a model, test its accuracy, and explore tools like ***Teachable Machine*** and ***Google Colab***. These tools help you build image, audio, or text-based ML models using simple steps. You can do these activities even on a basic computer with internet access.

5.1.1 Training a Model

Training means teaching the model to learn patterns from data. When you give the computer a set of examples with correct answers (called labels), it begins to understand how input features relate to output labels.

For example, if you show pictures of apples and bananas with their names, the model starts to learn the difference based on color, shape, and size.

5.1.2 Testing Accuracy

After the model is trained, it is tested on new data. The test data helps us understand how well the model can perform in real life.

Accuracy is calculated as:

Accuracy = (Correct Predictions ÷ Total Predictions) × 100

If your model gives correct answers most of the time, it is said to have high accuracy. If not, it needs better training data or more learning time.

5.2 Inputs, Outputs, and the Feedback Loop

5.2.1 Input Features

Features are the data that the model uses to make predictions. In different ML tasks, features can be:

- Numbers (e.g., hours studied)
- Images (e.g., hand gestures)
- Sounds (e.g., voice commands)

5.2.2 Output Labels

The model gives output in two ways:

- ***Classification:*** The output is a category or class. (Example: Cat or Dog)
- ***Regression:*** The output is a number. (Example: Predicting test marks)

5.2.3. Feedback Loop

ML models improve with feedback. If the model makes a wrong prediction, we tell it the correct answer. This is called feedback. When we retrain the model with this updated data, it becomes more accurate.

Steps in Feedback Loop:

1. Input is given to the model.
2. The model makes a prediction.
3. We check if the prediction is correct.
4. If wrong, correct the model.
5. Retrain the model to improve performance.

Observe the Figure 5.1 that shows input, output and feedback loops.

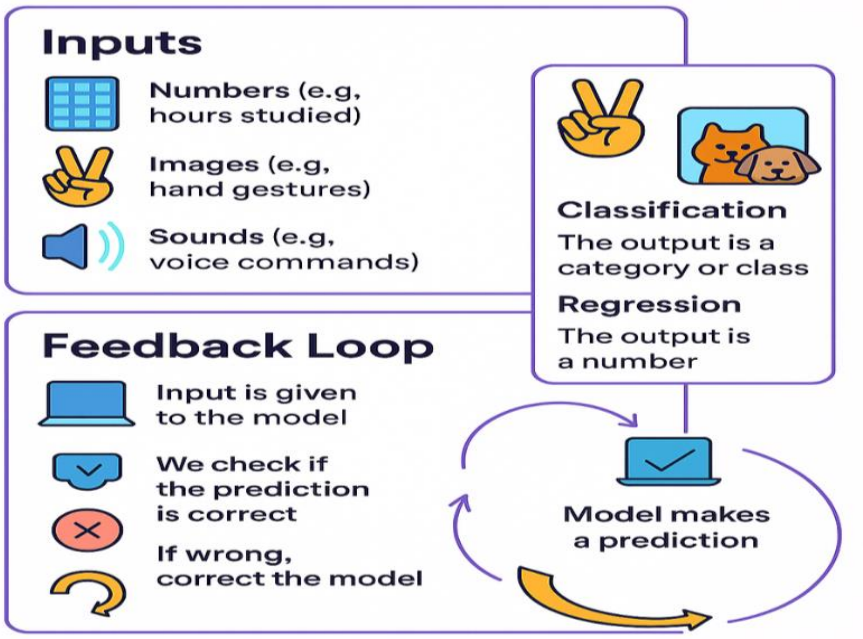


Fig. 5.1: Input, Output and Feedback loop

5.3 Introduction to ML Tools

5.3.1 Teachable Machine

Teachable Machine is a free, web-based tool by Google. It helps you build simple ML models using images, audio, or poses. You can train the model using your webcam or uploaded files.

Why use a Teachable Machine?

- No programming needed
- Works in a web browser
- Can export the model for offline use
- Helps you learn ML concepts easily

You can build a model in just 10–15 minutes.

Website: https://teachablemachine.withgoogle.com

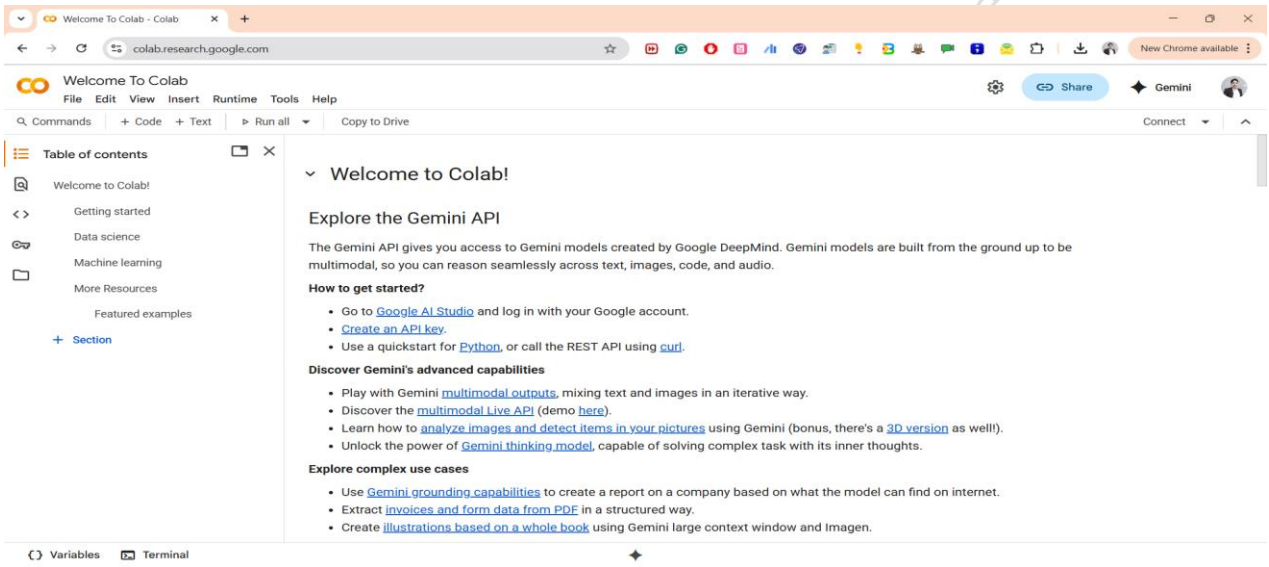
5.3.2 Google Colab

Google Colab is a free platform where you can run Python code in your browser. It is helpful for visual ML projects. Although it involves small amounts of code, you can copy and run pre-written code without learning programming in depth.

Why use Google Colab?

- Works without installing software
- Runs on low-end computers
- Good for visual learning
- Allows you to see real model results

Website: https://colab.research.google.com



5.4 Practical Activity: Image Classifier Using Teachable Machine

Objective: Build a simple image recognition model.

Step 1. Open the Teachable Machine website.

Step 2. Choose “Image Project”.

Step 3. Create two classes. Example: "Thumbs Up" and "Thumbs Down".

Step 4. Use your webcam or upload 10 images for each class.

Step 5. Click on “Train Model”. Wait for the model to learn.

Step 6. Test the model using your webcam. Show a thumbs up or thumbs down.

Step 7. Observe the prediction. Try to improve accuracy by adding more images.

Discussion Points:

- What feature is the model looking at?
- Are predictions correct?
- Does lighting or background affect the prediction?

5.5 Practical Activity: Regression Model on Google Colab

Objective: Use a pre-built model to understand prediction.

Steps:

Step 1. Go to Google Colab and open a new notebook.

Step 2. Copy and paste the following code:

```
from sklearn.linear_model import LinearRegression
import numpy as np
# Sample dataset
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2, 4, 6, 8, 10])  # y = 2 * x
model = LinearRegression().fit(X, y)
prediction = model.predict(np.array([[6]]))
print("Predicted value for 6:", prediction[0])
```

Run the code using Shift + Enter.

You will see the predicted value for input 6.

Discussion Points:

- What is the input feature?
- What does the model predict?
- How does it learn the pattern?

5.6 Visual Worksheet: The ML Pipeline

Create a simple diagram showing the following steps:

1. Input Features →
2. Model Training →
3. Output Prediction →
4. Feedback →
5. Retraining

Student Task: Fill in the blanks in the diagram and label each stage with its correct name.

This worksheet helps students understand how ML works step by step.

Summary

This session gives students practical exposure to ML using tools that don't require coding:

- Teachable Machine – a Google tool to build simple ML models using images, audio, or poses.
- Google Colab – a cloud-based platform for running Python code without installation. Students learn about training models, testing accuracy, and the role of features, labels, and the feedback loop in improving predictions. Activities include:

- Creating an image classifier with Teachable Machine (Thumbs Up vs. Thumbs Down).
- Running a regression model in Google Colab to predict values.
- Building a visual ML pipeline worksheet (Input → Training → Prediction → Feedback → Retraining).

This session shows that ML concepts can be learned through simple, interactive projects.

Check Your Progress

A. Multiple Choice Questions

1. What is a feature in ML? (a) Output of the model (b) Input data used for learning (c) A camera tool (d) A noise in audio
2. What does Teachable Machine allow you to create? (a) Code editor (b) Audio player (c) ML model using webcam or files (d) Antivirus program
3. What is the output of a regression model? (a) A number (b) A class name (c) An image (d) A sound clip
4. What tool is used to run small Python code online? (a) Notepad (b) MS-Paint (c) Google Colab (d) File Explorer
5. What does the feedback loop help with? (a) Deleting data (b) Slowing down training (c) Improving predictions (d) Opening a browser

B. Fill in the Blanks

1. Teachable Machine helps to build models using _____.
2. A _____ model is used to predict a number.
3. Google Colab is a tool to run _____ code online.
4. ML uses _____ and _____ to make predictions.
5. The _____ loop improves the model over time.

C. State Whether True or False

1. Teachable Machine works without writing any code.
2. Regression is used to predict labels like “Cat” or “Dog”.
3. A good dataset can improve model accuracy.
4. Google Colab runs Python code directly on your PC.
5. Features are the outputs of an ML model.

D. Short Answer Questions

1. What is meant by ‘training a model’?
2. List two tools used in this chapter for learning ML.
3. What is the main function of the feedback loop?
4. How do you create an image classifier in Teachable Machine?
5. Why is it helpful to use small datasets for learning?

Glossary

Module 1. Introduction to Artificial Intelligence

Accountability: Responsibility of developers or organizations for AI decisions and actions.

Actuators: Mechanical parts (motors, gears, arms) in robots that perform physical actions.

Algorithm: A step-by-step procedure for solving problems or performing tasks.

Amazon Alexa Skills Kit (ASK): A platform to develop voice-based applications ("skills") for Alexa-enabled devices.

Artificial General Intelligence (AGI): Hypothetical AI that can understand, learn, and perform any intellectual task like a human.

Artificial Intelligence (AI): Simulation of human intelligence in machines designed to think, learn, and make decisions.

Artificial Narrow Intelligence (ANI): AI specialized in a single task (e.g., Google Maps, ChatGPT).

Artificial Super Intelligence (ASI): A theoretical AI surpassing human intelligence in every domain.

Bias in AI: Errors in AI decision-making caused by unfair or unbalanced training data.

Chatbot: An AI tool that interacts with users through text or voice conversation.

Computer Vision (CV): A domain of AI that enables machines to process and understand images or videos.

Convolutional Neural Network (CNN): A deep learning model widely used in image and video recognition.

Data Analytics: Process of examining data to extract useful insights and patterns.

Decision Making: Process by which AI systems choose the best option from available alternatives.

Deep Learning: A subfield of machine learning using artificial neural networks with multiple layers.

Dialogflow: Google's AI platform for creating chatbots and virtual assistants.

Ethics in AI: Principles ensuring AI is used fairly, safely, and transparently.

Expert System: AI that mimics human expert decision-making using rules and knowledge bases.

Fairness: Ensuring AI does not discriminate on the basis of race, gender, or other attributes.

Framework (AI): A set of tools, libraries, and services to build and deploy AI models (e.g., TensorFlow).

Google Assistant: An AI-based virtual assistant that responds to user queries via text/voice.

IBM Watson Assistant: AI-powered chatbot and virtual agent platform for businesses.

Inference Engine: Component of an expert system that applies rules to data for decision-making.

Knowledge Base: A collection of facts and rules used by expert systems.

Limited Memory AI: AI that learns from past experiences but has no long-term memory (e.g., self-driving cars).

Machine Learning (ML): A branch of AI where machines learn from data without explicit programming.

Natural Language Processing (NLP): AI that enables machines to understand and respond to human language.

Neural Network: A computing model inspired by the human brain, used for learning from data.

Prediction: AI's ability to forecast outcomes based on historical data.

Reactive Machines: Basic AI that responds only to current input without memory (e.g., IBM Deep Blue).

Recommendation System: AI that suggests products or content based on user preferences (e.g., Netflix).

Reinforcement Learning: AI technique where systems learn through trial-and-error with rewards and penalties.

Robotics: Branch of AI that designs machines capable of sensing, thinking, and acting.

Self-aware AI: A hypothetical AI that possesses consciousness and self-understanding.

Sentiment Analysis: NLP technique for detecting emotions or opinions in text.

Supervised Learning: AI learning method using labeled training data.

Turing Test: A test to check if a machine's behavior is indistinguishable from human intelligence.

Transparency: The clarity and explainability of AI's decision-making process.

Unsupervised Learning: AI learning method where systems find patterns without labeled data.

Virtual Assistant: AI-powered assistant (e.g., Siri, Alexa, Google Assistant) that helps users via voice/text.

Module 2. Python Programming

Algorithm: A step-by-step procedure or set of rules to solve a problem or complete a task.

Boolean: A data type in Python representing two values: True or False.

Bug: An error or flaw in a program that causes incorrect results or unexpected behavior.

Comment: A line in Python code ignored by the interpreter, used for explanation/documentation (starts with #).

Compiler: A program that translates high-level programming code into machine language at once.

Data Type: Classification of data items, such as integer, float, string, or boolean.

Debugging: The process of identifying and fixing errors (bugs) in a program.

Float: A data type used to represent decimal numbers.

Function: A reusable block of code that performs a specific task.

Identifier: A name given to a variable, function, or class in Python.

Indentation: The spaces at the beginning of a line that define code blocks in Python.

Interpreter: A program that executes code line by line, as in Python.

Iteration: The process of repeating a set of instructions using loops.

Keyword: Reserved word in Python that has a predefined meaning and cannot be used as an identifier (e.g., if, while).

Library: A collection of pre-written functions and modules that can be used in programs.

Loop: A structure that allows repeated execution of a block of code (for and while loops).

Object: An instance of a class containing data and methods.

Operator: A symbol used to perform operations on variables and values (e.g., +, -, *).

Parameter: A variable used to pass information into a function.

Return Statement: A command in Python functions to send back a result to the caller.

String: A sequence of characters enclosed in quotes.

Syntax: The rules that define the correct structure of code in Python.

Tuple: An ordered, immutable collection of elements in Python.

Variable: A named memory location used to store data in a program.

Module 3. Data Literacy

Accuracy – How correct and reliable the data is compared to real-world values.

Algorithmic Bias – Bias that occurs when algorithms reflect flaws in training data.

Bias (in Data/AI) – Prejudice or unfairness in datasets or algorithms, often unintentional.

Big Data – Very large, complex datasets that require advanced tools for analysis.

Categorical Data – Data grouped into categories without numerical meaning (e.g., gender, city).

Completeness – Whether all required data is available without missing values.

Confidence Interval – Range that estimates population values with a certain level of confidence (e.g., 95%).

Consistency – Uniformity of data across records and systems.

Data – Raw, unprocessed facts and figures without context.

Data Cleaning – Process of fixing or removing incorrect, incomplete, or duplicate data.

Data Collection Methods – Ways to gather data (e.g., surveys, interviews, observation, sensors, web scraping, experiments).

Data Monopolies – Situation where few companies control vast amounts of data.

Data Quality – How accurate, complete, and consistent data is for use.

Data Storytelling – Combining data, visuals, and narrative to communicate insights.

Data Visualization – Representation of data using charts and graphs.

Descriptive Statistics – Methods to summarize and describe data (mean, median, mode, range).

Digital Economy – Economic activities powered by digital technologies, data, and online interactions.

Duplicate Records – Repeated entries of the same data.

Ethics in Data – Principles ensuring privacy, transparency, security, and accountability in data use.

Histogram – Graph showing distribution of data across intervals.

Hypothesis Testing – Method to test if an assumption about a dataset is correct.

Information – Processed data that has meaning and is useful for decision-making.

Inferential Statistics – Using a sample to make predictions about a larger population.

Mean (Average) – Sum of values divided by number of values.

Median – Middle value in an ordered dataset.

Mode – Most frequently occurring value in a dataset.

Numerical Data – Data represented by numbers (discrete or continuous).

Open Data – Publicly available data free to use and share (e.g., census, weather data).

Ordinal Data – Data with ordered categories (e.g., ranks: 1st, 2nd, 3rd).

Outliers – Extreme or abnormal values that don't match real-world expectations.

Qualitative Data – Descriptive data about qualities or opinions (e.g., color, feedback).

Quantitative Data – Numerical data that can be measured (e.g., height, marks).

Range – Difference between maximum and minimum values.

Real-Time Data – Data collected and processed instantly as events happen (e.g., Uber location tracking).

Regression Analysis – Statistical method to predict outcomes based on data.

Sampling Bias – Error from collecting data from a non-representative group.

Standard Deviation – Measure of how spread out data is from the mean.

Standardization – Converting data into a uniform format (e.g., date format, units).

Statistics – Science of collecting, analyzing, and interpreting data.

Structured Data – Organized data stored in tables, rows, and columns (e.g., databases, spreadsheets).

Textual Data – Data in text form (e.g., reviews, comments).

Transparency – Openness about how data is collected and used.

Typographical Error (Typo) – Mistakes in typing data (e.g., "Rhaul" instead of "Rahul").

Unstructured Data – Data without a fixed format (e.g., videos, images, messages).

Validation – Checking if data follows predefined rules (e.g., age between 0–120).

Module 4. Mathematics for AI

Adjacency Matrix – A matrix representing connections between graph nodes.

Back-propagation – Algorithm that uses the chain rule to adjust weights in neural networks.

Bayes' Theorem – Formula that updates probabilities based on new evidence.

Bijective Function – Function that is both injective (one-to-one) and surjective (onto).

Boolean Algebra – Mathematical structure for logical operations using 0 (False) and 1 (True).

Cartesian Product (A × B) – Set of ordered pairs from two sets.

Chain Rule – A rule to differentiate composite functions, crucial in neural networks.

Combinatorics – Branch of mathematics dealing with counting and arrangements.

Conditional Probability – Probability of one event given that another has already occurred.

Cost Function (Loss Function) – A function that measures the error between predicted and actual values.

Covariance – Measure of how two variables change together.

Decision Tree – A tree-structured model used for classification and decision-making.

Derivative – The instantaneous rate of change of a function; slope of the tangent line.

Difference (A – B) – Elements in set A but not in set B.

Eigenvalue – A scalar that shows how much the eigenvector is stretched or compressed.

Eigenvector – A vector that does not change direction under a linear transformation.

Empirical Probability – Probability based on experimental data or past observations.

Expected Value (Expectation) – The average outcome if an experiment is repeated many times.

Function – A relation that maps each input to exactly one output.

Gradient – A vector of all partial derivatives; points in the direction of steepest increase.

Gradient Descent – Optimization algorithm that updates weights in the opposite direction of the gradient to minimize error.

Graph – A structure made of nodes (vertices) and edges (connections).

Intersection (A ∩ B) – Set of elements common to both sets.

Injective Function (One-to-One) – Each input maps to a unique output.

Learning Rate (η) – A parameter that controls the size of steps in gradient descent.

Limit – The value a function approaches as the input gets closer to a certain point.

Linear Transformation – A function that maps vectors to other vectors while preserving vector operations.

Logic Gates – Digital circuits (AND, OR, NOT, etc.) that implement logical operations.

Matrix – A rectangular array of numbers arranged in rows and columns, used to store data or transformations.

Matrix Multiplication – Operation combining rows of one matrix with columns of another, used in neural networks.

Modulus Function – A function that gives the absolute value of the input, often used in ML loss functions.

Normal Distribution (Gaussian) – A bell-shaped distribution used widely in AI.

Optimization – Process of finding the best (minimum or maximum) value of a function.

Partial Derivative – Derivative of a multi-variable function with respect to one variable, keeping others constant.

Power Set – Set of all subsets of a set.

Predicate Logic (FOL) – Logic with variables and quantifiers (∀ = for all, ∃ = there exists).

Probability – Measure (between 0 and 1) of how likely an event is to occur.

Probability Distribution – A function that shows probabilities of all possible values of a random variable.

Propositional Logic – Logic dealing with statements that are either true or false .

Random Variable – A variable whose value depends on the outcome of a random experiment.

Recursion – A process where a function calls itself to solve smaller sub-problems .

Rotation – Transformation that turns a shape around a point.

Scaling – Transformation that enlarges or shrinks a shape.

Set – A collection of distinct objects.

Shearing – Transformation that slants the shape of an object.

Subjective Probability – Probability based on personal judgment or experience.

Surjective Function (Onto) – Every output has at least one input.

Tensor – A generalization of vectors and matrices to higher dimensions (e.g., images, videos).

Theoretical Probability – Probability based on logic and formulas, assuming equal chances.

Transpose – Flipping a matrix across its diagonal (rows ↔ columns).

Union (A ∪ B) – Set of all elements in A, B, or both.

Variance – Measure of how spread out data values are around the mean.

Vector – An ordered list of numbers that represent magnitude and direction.

Module 5. Fundamentals of Machine Learning

Algorithm – A step-by-step procedure or set of rules used by computers to solve a problem or perform a task.

Bias (in AI/ML) – Systematic error that causes unfair or incorrect outcomes due to unbalanced or flawed data.

Classification – A type of supervised learning where the goal is to predict discrete categories (e.g., cat or dog).

Clustering – An unsupervised learning method that groups data points into clusters based on similarity.

Data Pre-processing – Cleaning and transforming raw data to make it suitable for training machine learning models.

Dataset – A collection of data used to train, test, and evaluate a machine learning model.

Deep Learning (DL) – A subset of machine learning that uses deep neural networks with many layers to learn complex patterns automatically.

Evaluation – The process of testing a trained model on new, unseen data to measure its performance.

Feature(s) – Input variables (or attributes) fed into a machine learning model to help make predictions.

Feature Extraction – The process of identifying and selecting relevant features from raw data for model training.

Feedback Loop – A process where the output of a system is used to improve its future predictions or performance.

Label – The output or answer that a machine learning model is trained to predict (e.g., "Pass"/"Fail").

Model – A mathematical representation of a real-world process that learns patterns from data to make predictions.

Prediction – The output generated by a trained machine learning model for new, unseen input data.

Regression – A type of supervised learning used to predict numerical values (e.g., house prices).

Reinforcement Learning (RL) – A type of machine learning where an agent learns by interacting with an environment and receiving rewards or penalties.

Supervised Learning – Machine learning method where the model is trained on labelled data (inputs with correct outputs).

Testing – The stage of evaluating a trained model's performance using data that was not part of training.

Training – The process of feeding data into a machine learning algorithm so the model can learn patterns.

Unsupervised Learning – A type of machine learning where the model finds hidden patterns or groupings in unlabelled data.

Answer Key

Module 1. Introduction to Artificial Intelligence (AI)

Session 1. Concept of AI

A. Multiple Choice Questions

1. (b) 2. (b) 3. (b) 4. (b) 5. (d)

B. Fill in the blanks

1. human 2. ELIZA 3. Deep Blue 4. AlphaGo 5. algorithms

C. True or False

1. False 2. True 3. False 4. True 5. False

Session 2. Types of AI

A. Multiple Choice Questions

1. (c) 2. (b) 3. (c) 4. (b) 5. (b)

B. Fill in the blanks

1. ASI 2. Reactive 3. Unsupervised 4. Reinforcement 5. neural

C. True or False

1. True 2. False 3. True 4. True 5. False

Session 3. Domains of AI

A. Multiple Choice Questions

1. (b) 2. (c) 3. (c) 4. (c) 5. (b)

B. Fill in the blanks

1. Natural Language Processing 2. Computer Vision 3. Expert 4. Robotics 5. PyTorch

C. True or False

1. True 2. False 3. True 4. False 5. True

Module 2. Python Programming

Session 1. Python Fundamentals

A. Multiple Choice Questions

1. (d) 2. (c) 3. (c) 4. (b) 5. (b) 6. (c) 7. (b) 8. (b) 9. (c) 10. (b) 11. (c) 12 (a) 13. (b) 14. (a) 15. (b) 16. (c) 17. (c) 18. (c) 19. (c) 20. (b) 21. (c) 22. (b) 23. (b) 24. (a) 25. (b)

B. Fill in the Blanks

1. py 2. = 3. identifiers 4. case 5. input() 6. string 7. int() 8. print() 9. # 10. % 11. ** 12. assignment 13. continue 14. expression 15. unlimited 16. keywords 17. explicit, casting 18. multiplication, addition 19. False 20. True 21. int() 22. assignment 23. input() 24. implicit, explicit 25. ValueError

C. State whether True or False

1. False 2. False 3. False 4. True 5. True 6. False 7. True 8. False 9. True 10. False

Session 2. Control Flow in Python

A. Multiple Choice Questions

1. (c) 2. (b) 3. (b) 4. (b) 5. (c) 6. (c) 7. (c) 8. (b) 9. (b) 10. (b) 11. (a) 12. (d) 13. (a) 14. (b) 15. (b)

B. Fill in the Blanks

1. flow 4. repetition 3. sequential 4. conditional 5. if 6. else 7. if-elif-else 8. loops 9. for 10. while 11. break 12. continue 13. pass 14. indented 15. flow

C. State whether True or False

1. True 2. False 3. False 4. True 5. False 6. True 7. True 8. False 9. True 10. True

Session 3. Python Data Types

A. Multiple Choice Questions

1. (d) 2. (c) 3. (c) 4. (c) 5. (c) 6. (d) 7. (b) 8. (d) 9. (b) 10. (e) 11. (c) 12. (c) 13. (a) 14. (b) 15. (a) 16. (c) 17. (e) 18. (b) 19. (c) 20. (a) 21. (c) 22. (e) 23. (b) 24. (d) 25. (b)

B. Fill in the Blanks

1. single, double 2. mutable 3. immutable 4. key-value 5. immutable 6. unique 7. mutable 8. immutable 9. lists, dictionaries, and sets 10. strings, tuples, and numbers (integers, floats) 11. data type 12. len() 13. slice 14. TypeError 15. end 16. end 17. specified index 18. first occurrence 19. list 20. all 21. value 22. slicing ([:]) 23. different 24. unique, immutable 25. key 26. KeyError 27. key-value tuples 28. concatenate 29. indexing 30. substring

C. State whether True or False

1. False 2. True 3. True 4. True 5. False 6. True 7. False 8. False 9. True 10. False 11. False 12. True 13. True 14. True 15. True 16. False 17. True 18. True 19. False 20. False

Session 4. Python Functions

A. Multiple Choice Questions

1. (b) 2. (b) 3. (b) 4. (c) 5. (c) 6. (b) 7. (c) 8. (c) 9. (d) 10. (d) 11. (d) 12. (a)

B. Fill in the Blanks

1. reusable 2. def 3. units, reuse 4. arguments 5. None 6. local 7. return 8. global 9. arguments 10. method

C. State whether True or False

1. False 2. False 3. True 4. False 5. False 6. False 7. False 8. True 9. True 10. False 11. False 12. True 13. False 14. True 15. False

Module 3. Data Literacy

Session 1. Role of Data in Society

A. Multiple Choice Questions

1. (b) 2. (c) 3. (b) 4. (c) 5. (a)

B. Fill in the Blanks

1. Data, Information 2. Cloud 3. Data 4. Bias 5. Ethical

C. True or False

1. True 2. False 3. True 4. False 5. True

Session 2. Types and Sources of Data

A. Multiple Choice Question

1. (c) 2. (b) 3. (b) 4. (d) 5. (d)

B. Fill in the Blanks

1. Structured, Unstructured 2. Qualitative 3. Textual 4. Open 5. Sensors

C. True or False

1. True 2. False 3. True 4. False 5. True

Session 3. Data Quality and Data Cleaning

A. Multiple Choice Question

1. (d) 2. (b) 3. (b) 4. (d) 5. (c)

B. Fill in the Blanks

1. Consistency 2. Missing 3. Data cleaning 4. Formats 5. Decisions

C. True or False

1. False 2. True 3. True 4. False 5. True

Session 4. Statistics for Data Analysis

A. Multiple Choice Questions

1. (c) 2. (a) 3. (b) 4. (b) 5. (c)

B. Fill in the Blanks

1. Mode 2. Maximum 3. Visualization 3. Narrative 5. Sample

C. True or False

1. False 2. True 3. True 4. False 5. True

Module 4. Mathematics for AI

Session 1. Linear Algebra Concepts Used in AI

A. Multiple Choice Questions

1. (b) 2. (c) 3. (b) 4. (a) 5. (b)

B. Fill in the Blanks

1. Vector 2. Square 3. Dot 4. Square 5. Matrix

C. True or False

1. False 2. True 3. True 4. False 5. True

Session 2. Calculus for Optimization

A. Multiple Choice Questions

1. (a) 2. (b) 3. (b) 4. (c) 5. (a)

B. Fill in the Blanks

1. Differentiation 2. Integration 3. Opposite 4. Saddle 5. Zero

C. True or False

1. True 2. True 3. True 4. False 5. False

Session 3. Probability and Statistics in AI

A. Multiple Choice Questions

1. (c) 2. (a) 3. (b) 4. (c) 5. (b)

B. Fill in the Blanks

1. 0, 1 2. Mode 3. Mean 4. Conditional 5. Variance

C. True or False

1. True 2. False 3. True 4. True 5. False

Session 4. Discrete Mathematics for AI

A. Multiple Choice Questions

1. (a) 2. (d) 3. (b) 4. (b) 5. (a)

B. Fill in the Blanks

1. Set 2. Proposition 3. True, False 4. Graph 5. 2n

C. True or False

1. False 2. True 3. True 4. False 5. True

Module 5. Fundamentals of Machine Learning

Session 1. Introduction to Machine Learning

A. Multiple Choice Questions

1. (c) 2. (b) 3. (b) 4. (b) 5. (b)

B. Fill in the Blanks

1. Machine Learning model / algorithm 2. Label / output 3. Preprocessing / cleaning 4. Evaluation / testing 5. Explicitly

C. True or False

1. True 2. False 3. True 4. False 5. True

Session 2. Types of Machine Learning

A. Multiple Choice Questions

1. (c) 2. (b) 3. (b) 4. (c) 5. (b)

B. Fill in the Blanks

1. Supervised 2. Unsupervised 3. Reward 4. Classification 5. Unsupervised

C. True or False

1. True 2. False 3. True 4. False 5. True

Session 3. Data in Machine Learning

A. Multiple Choice Questions

1. (c) 2. (b) 3. (c) 4. (c) 5. (c)

B. Fill in the Blanks

1. Features 2. Label 3. Training 4. Accurate 5. Clean

C. True or False

1. True 2. False 3. False 4. True 5. True

Session 4. Ethics and Responsibility in AI/ML

A. Multiple Choice Questions

1. (c) 2. (c) 3. (b) 4. (b) 5. (b)

B. Fill in the Blanks

1. Features 2. Label 3. Training 4. Accurate 5. Clean

C. True or False

1. True 2. False 3. True 4. False 5. False

Session 5. Hands-On ML Without Code (Using Tools)

A. Multiple Choice Questions

1. (b) 2. (c) 3. (a) 4. (c) 5. (c)

B. Fill in the Blanks

1. Webcam or files 2. Regression 3. Python 4. Data and algorithms 5. Feedback

C. True or False

1. True 2. False 3. True 4. False 5. False