Python For All

Course Code: SGB24CA202SE

Skill Enhancement Course

Four Year Undergraduate Programmes

Self Learning Material

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University of Education, Training and Research in Blended Format, Kerala

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Python For All

Course Code: SGB24CA202SE

Semester - III

Skill Enhancement Course
For FYUG Programmes (Honours)
Self Learning Material

(With Model Question Paper Sets)

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

Python For All

Course Code: SGB24CA202SE
Semester- III
Skill Enhancement Course
Four Year Undergraguate Programmes

Scan this QR Code for reading the SLM on a digital device.

Edition:
October 2025

ISBN 978-81-987966-7-7

www.sgou.ac.m

Visit and Subscribe our Social Media Platforms

Message from Vice Chancellor

Dear Learner,

It is with great pleasure that I welcome you to the Four Year UG Programme offered by Sreenarayanaguru Open University.

Established in September 2020, our university aims to provide high-quality higher education through open and distance learning. Our guiding principle, 'access and quality define equity', shapes our approach to education. We are committed to maintaining the highest standards in our academic offerings.

Our university proudly bears the name of Sreenarayanaguru, a prominent Renaissance thinker of modern India. His philosophy of social reform and educational empowerment serves as a constant reminder of our dedication to excellence in all our academic pursuits.

The University is committed to fostering a future-ready learning environment that emphasizes both knowledge and practical skill development. As part of the FYUG programme, the Skill Enhancement Elective Course "Python for All" introduces learners to one of the most versatile and widely used programming languages in the world. Designed for learners from all academic disciplines, this course provides a hands-on introduction to Python, focusing on its core syntax, logical problem-solving techniques, and real-world applications. Through engaging exercises and practical examples, you will learn how Python can be applied across diverse fields such as data analysis, automation, web development, and scientific research. The course aims to build your confidence in computational thinking and digital literacy—skills that are essential in today's technology-driven world—while supporting interdisciplinary learning and personal growth throughout your academic journey.

Our teaching methodology combines three key elements: Self Learning Material, Classroom Counselling, and Virtual modes. This blended approach aims to provide a rich and engaging learning experience, overcoming the limitations often associated with distance education. We are confident that this programme will enhance your understanding of statistical methods in business contexts, preparing you for various career paths and further academic pursuits.

Our learner support services are always available to address any concerns you may have during your time with us. We encourage you to reach out with any questions or feedback regarding the programme.

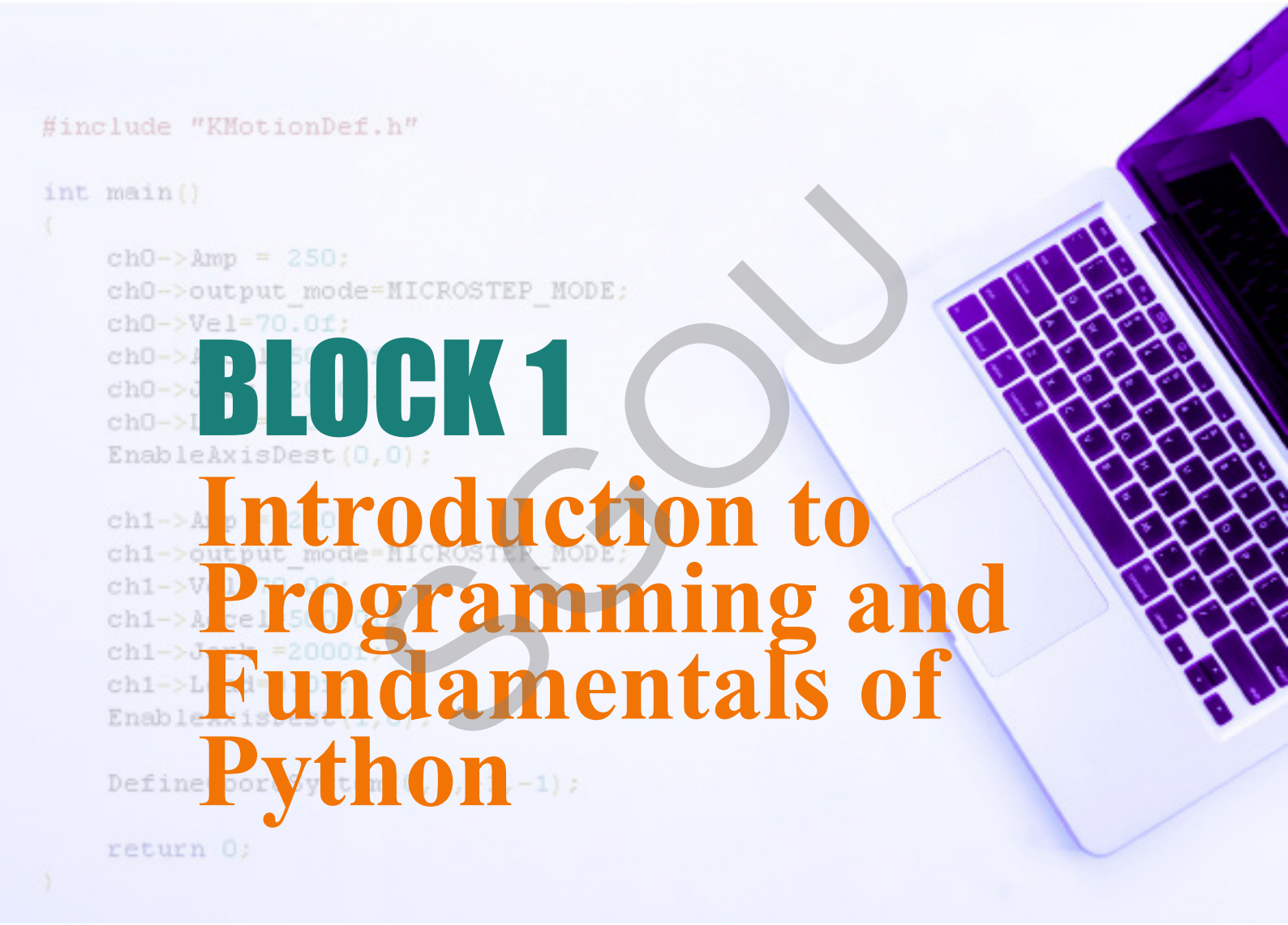
We wish you success in your academic journey with Sreenarayanaguru Open University.

Best regards,

Dr. Jagathy Raj V.P.
Vice Chancellor

01-10-2025

Contents



BLOCK 1

Introduction to Programming and Fundamentals of Python

Introduction to Computing and Concepts of Programming

Learning Outcomes

At the end of this unit, the learner will be able to:

- define a computer system and its basic components.
- list the different types of computer memory.
- identify common input and output devices.
- recall the basic building blocks of a computer system.
- familiarise different types of programming languages and translators.

Prerequisites

Have you ever used a calculator to solve a math problem or typed something into your phone and got a result instantly? These everyday actions show how computers work by following a set of clear instructions to perform tasks. Behind the scenes, a computer takes input, processes the data, stores it if needed, and gives you the correct output. Just like our body needs different organs to work together, a computer system uses hardware and software to complete its tasks. Also, just like we plan before doing something, such as making a shopping list before going to the store, computer programs are also created through a proper step-by-step process. This is the phases of programming, where programmers define the problem, plan the steps, write code, check for mistakes, and test the program to make sure it works correctly. In this lesson, you will explore the basic building blocks of a computer system and how programming helps it solve real-world problems efficiently.

Keywords

Data Processing, Compiler, Assembler, Interpreter, Algorithm, Flowchart

Discussion

A computer system is a digital machine that takes input, processes the data, stores it, and generates output. It operates by executing a set of instructions known as Programs, allowing it to perform various tasks efficiently and accurately. The system is made up of two main parts: hardware, which refers to the physical components, and software, which includes the data and the instructions. From handling basic calculations to managing advanced scientific tasks, computers are vital in many areas of modern life such as education, communication, business, and research.

1.1.1 Basic Building Blocks

For a computer system to function, the core elements required are shown in Fig 1.1.1.

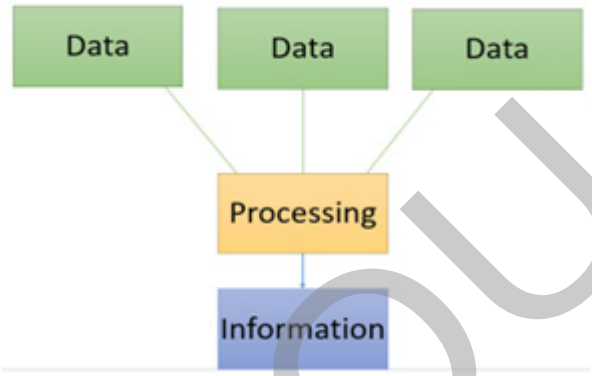


Fig 1.1.1 Building Blocks

1.1.1.1 Data

Imagine you need to calculate the sum of 3 and 4. You input "3+4" into a computer. In this scenario, the numbers 3 and 4 are considered data.

Data is a collection of facts in a raw or unorganized form such as numbers, characters, special characters, image, or audio. Data is represented with the help of characters such as alphabets (A-Z, a-z), digits (0-9) or special characters (+, -, /, *, <,>, = etc.).

1.1.1.2 Information

The computer processes the above data by interpreting the operation "3+4" and performing the addition, resulting in the output of 7. Information is organized or classified data, which has some meaningful values for the receiver. Information is the processed data on which decisions and actions are based.

1.1.1.3 Data Processing

The process of converting data into information is called Data Processing

Example: The provided data, '4th,' 'girl,' 'studying,' 'Diya,' '9,' 'old,' and 'standard,' can be transformed into meaningful information.

'Diya, a 9-year-old girl, is studying in the 4th standard".

1.1.2 Components of a Computer System

Just as the human body relies on various organs such as the brain, heart, and lungs working together to maintain life and perform daily activities, a computer system also depends on its different components functioning in harmony. A computer system mainly includes a central processing unit (CPU), memory, input/output devices, and storage. Fig 1.1.2 illustrates the block diagram of a computer system. The arrows indicate how data and signals move between the different components.

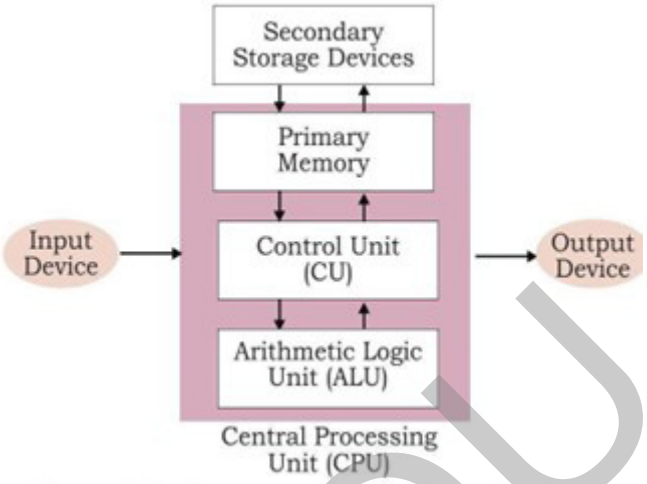


Fig 1.1.2 Computer System Components

1.1.2.1 Input Devices

Input devices are electronic tools that receive data or instructions from the external environment and convert them into a format that the computer can understand. These devices act as a bridge between the user and the computer, enabling smooth interaction. When users enter data through input devices, that data is stored in the computer's memory for later processing and use. Once the processing is done, the final output can be retrieved using output devices. In essence, input devices send information to the computer, allowing users to interact with and manage the system.

Fig 1.1.3 Input Devices

1.1.2.2 Central Processing Unit (CPU)

The CPU, often called the Brain of the Computer, is the electronic component

responsible for carrying out actual data processing. Also referred to as the Processor or Microprocessor, it is physically located on one or more integrated circuits (ICs) made from semiconductor materials. The CPU receives instructions and data through software programs. It retrieves this information from memory, executes arithmetic and logical operations based on the instructions, and then returns the results to memory.

Three subsystems together form a central processing unit.

- **Arithmetic and Logical Unit (ALU):** ALU is responsible for performing arithmetic and logical operations.
- **Control Unit (CU):** Responsible for managing the execution of instructions and coordinating the various operations of the processor.
- **Registers:** Register refers to a small, fast storage location within the central processing unit (CPU) of a computer. Registers are used to store instructions, data and intermediate results that are currently being processed by the CPU.

1.1.2.3 Memory Unit

Memory is used to store data either permanently, for future use, or temporarily, at the time of processing. Processed or unprocessed data can be stored in computer memory. For processing, the computer takes data from memory. After processing, the computer stores data into the computer memory for future use. Computer memory is of two types

- Primary Memory
- Secondary Memory

1. Primary Memory

Primary memory or the Main memory is a temporary memory, which stores data that is currently executing on the CPU. The CPU can only access data from primary memory not from the secondary storage. Two primary memories are RAM and ROM

Random Access Memory (RAM): RAM is volatile memory whose contents are erased when the system's power is turned off or interrupted. RAM is the main memory of the computer used to store data that the CPU is currently working with or needs to access quickly. The more RAM a computer has, the more programs and data it can work with simultaneously without slowing down. Two types of RAM are Static RAM and Dynamic RAM.

Read Only Memory (ROM): ROM is a non-volatile memory that retains contents even after the power is turned off or interrupted by the computer system. It is a read only memory as we can only read the programs and data stored on it but cannot write on it. ROM stores data permanently. ROM primarily stores boot instructions. Different types of ROM are PROM, EPROM, EEPROM, Flash ROM.

2. Secondary Memory

Secondary memory is also known as Auxiliary Memory or External Memory stores data and instruction permanently for later processing. It is a non-volatile memory device. It is a bulk storage device. The CPU cannot directly access the secondary memory.

Commonly used secondary memory are: Hard disk, Flash memory, CD, DVD, Blu-ray disc etc. Figure 1.1.4 shows different types of secondary memory.

Fig 1.1.4 Secondary Devices

1.1.2.4 Output Devices

An output device is a hardware component that displays or presents the processed data from a computer to the user in various forms such as text, audio, video, visuals on a screen, or printed copies on paper. These devices are generally classified into categories like audio output devices, visual output devices, audio-visual output devices, and print-based output devices. Depending on the type of output and user needs, different output devices can be connected to a computer system to deliver the final results. Common examples of output devices include monitors, graphic plotters, printers, speakers, headphones. Each of these devices serves a specific purpose in presenting information in a user-friendly format. Fig 1.1.5 shows common output devices.

Fig 1.1.5 Output Devices

1.1.3 Computer Language

A language serves as a means of communication, enabling the exchange of information, ideas, and opinions. Similarly, in computer systems, programming languages are used by software developers to build applications and software solutions. These languages offer a structured way to write instructions that allow a computer to carry out specific tasks. Some common programming languages include C, C++, Java, Python, Ruby, Perl.

Programming languages are categorized into two main types based on how closely they

interact with the computer's hardware as

- Low-Level Languages
- High-Level Languages

1.1.3.1 Low Level Languages

A low-level programming language is closer to the machine's architecture and provides a higher level of control over hardware resources. Machine language and assembly language are low level languages.

- **Machine Language:** Also known as Binary Language. It is the only language that can be understood by the computer. That is the language consisting of only 0's and 1's. It is easy and fast to execute but programmers can find it difficult to understand and code.
- **Assembly Language:** This language uses some mnemonics for writing programs. It also needs to be translated into machine language.

1.1.3.2 High Level Languages

High-level language is closer to human languages. It is written in English like languages. It needs to be translated into machine language. Examples for high level languages are C++, Java, Python etc.

The following table 1.1.1 highlights all the major differences between high-level language and low-level language.

Table 1.1.1 Differences between high-level language and low-level language

Low Level	High Level
Considered as a machine-friendly language	Designed to be user-friendly and easy for programmers to use.
Requires an assembler to translate instructions into machine code.	Needs a compiler or interpreter to convert into machine-readable form.
Debugging is more complicated and time-consuming	Troubleshooting and fixing errors is straightforward.
More efficient in terms of memory and performance.	Uses more memory and system resources.

1.1.4 Language Translators

Computer programs are usually developed using high-level programming languages such as C++, Python, and Java. A language translator, also known as a Language Processor, is a type of software that transforms Source Code written in one programming language into another language or directly into machine code (also called Object Code).

During this process, it also detects and reports any errors in the code.

1.1.4.1 Types of Language Translators

1. Compiler

A compiler is a type of language processor that reads the entire source code written in a high-level language at once and converts it into an equivalent machine language program as shown in Fig 1.1.6. Examples of compiled languages include C, C++, and C#.

In the compilation process, the source code is only translated into object code if it contains no errors. If errors are present, the compiler lists them along with their corresponding line numbers after the compilation is complete. These errors must be corrected before the compiler can successfully translate the code. Once compiled without errors, the resulting object code can be executed multiple times without needing to recompile.

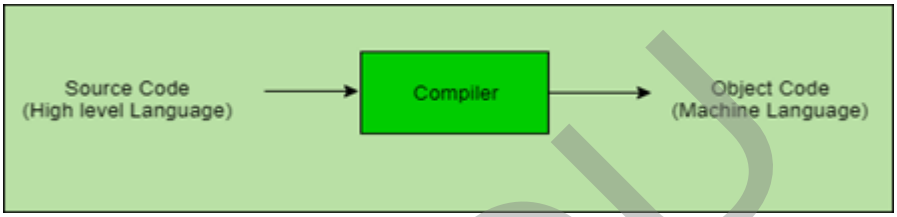


Fig 1.1.6 Compiler

2. Assembler

An assembler is used to convert programs written in assembly language into machine code as in Fig 1.1.7**.** It takes the assembly language source code as input, which consists of low-level instructions, and produces object code or machine code as output, which the computer can execute. Assembly language code consists of mnemonic instructions such as ADD, MUL, SUB, DIV, MOV, MUX, and others, which represent low-level operations.

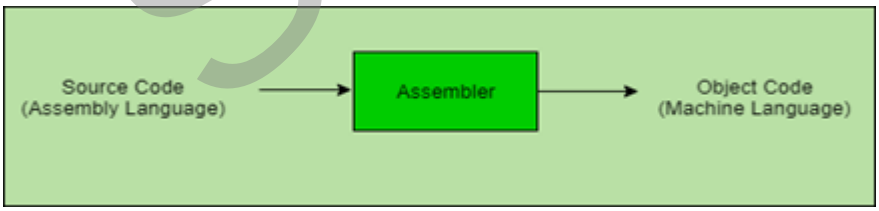


Fig 1.1.7 Assembler

3. Interpreter

An interpreter is a language processor that translates and executes each line of a source program one at a time. If it encounters an error in a statement, it stops the translation process at that point and displays an error message. Execution continues to the next line only after the error is corrected. Unlike a compiler, an interpreter does not convert the entire source code into object code beforehand; instead, it directly executes the instructions line by line as in Fig 1.1.8.

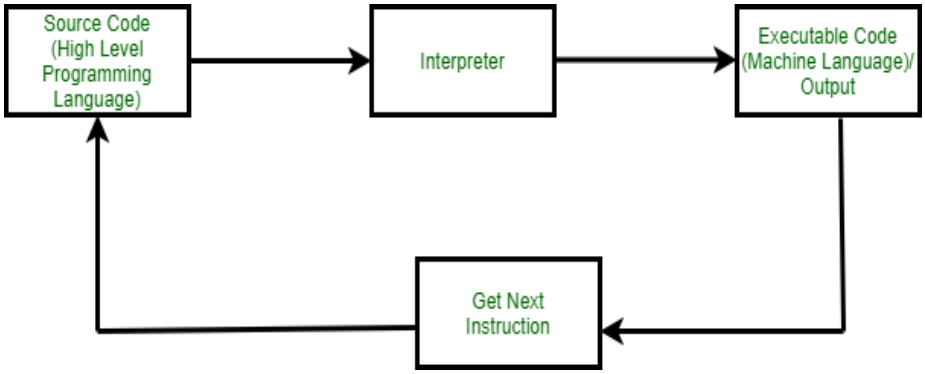


Fig 1.1.8 Interpreter

1.1.5 Simple Algorithms and Flowcharts

In programming, algorithms and flowcharts are used to break down complicated problems into simpler, more manageable steps, making it easier to analyze and find effective solutions.

1.1.5.1 Algorithm

To successfully complete tasks in our daily lives, having a plan or step-by-step procedure is essential. For example, when preparing an omelet, there is a specific procedure to follow

- Step 1: Crack open an egg
- Step 2: Dice the onion into small pieces
- Step 3: Chop the green chili
- Step 4: Add salt to taste
- Step 5: Mix the ingredients thoroughly
- Step 6: Heat the pan
- Step 7: Pour the prepared mixture into the heated pan
- Step 8: The omelet is ready to be served

An algorithm is a step-by-step set of instructions, or a sequence of well-defined computational steps designed to perform a specific task or solve a particular problem. The key characteristics of an algorithm include:

1. **Input:** Algorithms take input, which is the data or information the algorithm will process
2. **Output:** Algorithms produce output, which is the result or solution generated after processing the input.

3. **Definiteness:** Each step of the algorithm must be precisely and unambiguously defined.

4. **Finiteness:** An algorithm must have a finite number of steps, meaning it should eventually stop and produce a result.

5. **Effectiveness:** The steps in an algorithm should be executable, and each step should be feasible to carry out using available resources.

6. **Uniqueness:** An algorithm should lead to a unique solution for a given input.

Examples for Algorithm

1. Write an algorithm to find sum of three numbers

   Step 1: Start

   Step 2: Read a, b, c

   Step 3: Sum = a+b+c

   Step 4: Print sum

   Step 5: Stop

2. Write an algorithm to find the largest among two numbers.

   Step 1: Start

   Step 2: Read a, b

   Step 3: If (a>b) go to Step 4 else go to Step 5

   Step 4: Print “a is greater than b” go to Step 6

   Step 5: Print “b is greater than a”

   Step 6: Stop

3. Write an algorithm to find the factorial of a number

   Step 1: Start

   Step 2: Read n

   Step 3: fact = 1

   Step 4: If (n >= 1) then go to Step 4 else go to Step 6

   Step 5: fact = fact*n

   Step 6: n = n-1 go to Step 3

   Step 7: Print fact

Step 8: Stop

1.1.5.2 Flowchart

Flowchart is the pictorial representation of an algorithm, using standardized symbols and shapes to illustrate the logical steps and decision-making processes in a sequential manner. Each symbol in a flowchart has a specific meaning, making it easier to understand the flow of the algorithm.

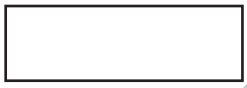
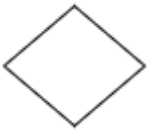
Types of Flowchart

- **System Flowchart:** shows the processing of the entire system. It describes the input/output devices, the media being used and the flow of data in the system.
- **Program flowchart:** shows the complete steps involved in the execution of a program including I/O, processing, loops, and branching. It is more detailed than the system flowchart.

Common Flowchart Symbols

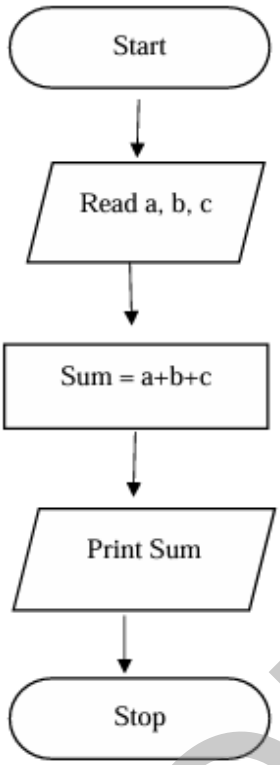
Table 1.1.2 shows the common symbols used in Flowchart representation. Using these symbols, you can create a visual representation of the steps and decisions in your algorithm, making it easier for others to understand and for you to identify potential issues or improvements

Table 1.1.2 Flowchart Symbols

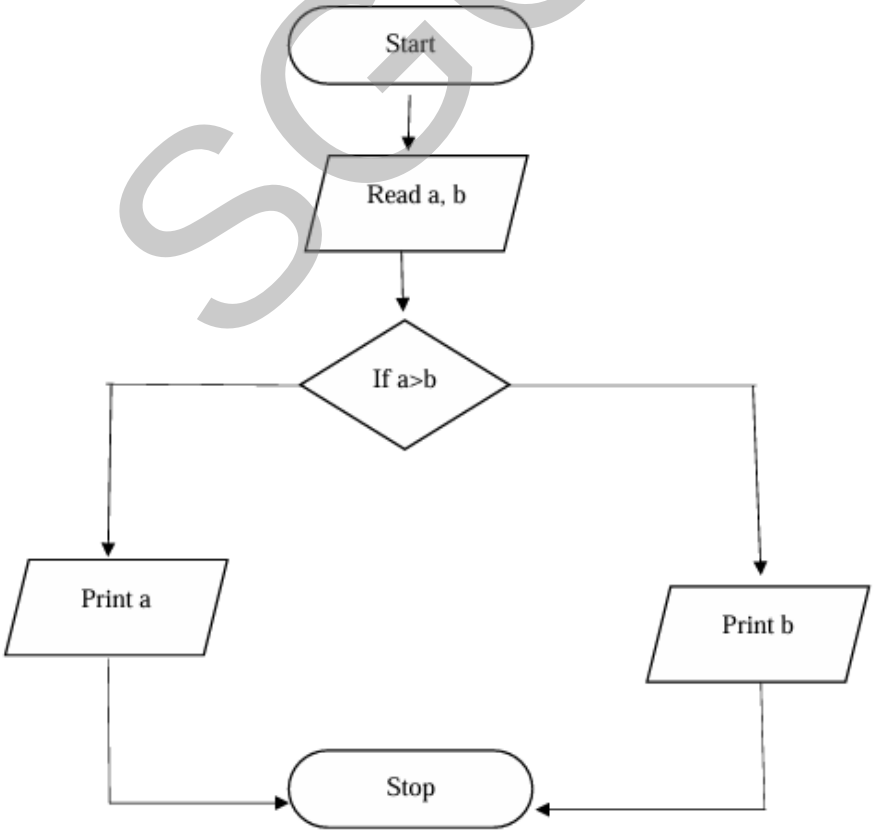
Symbol	Name	Function
	Start/End	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

Examples

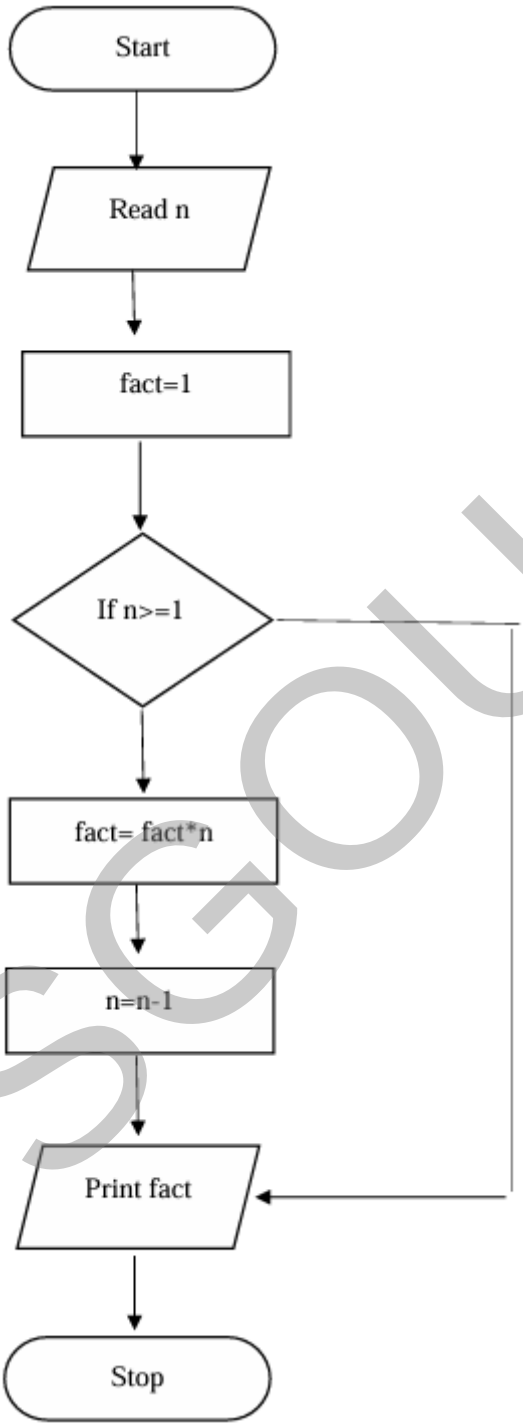
1. Draw a flowchart to find sum of three numbers.



2. Draw a flowchart to find the largest among two numbers.



3. Draw a flowchart to find the factorial of a number.



1.1.6 Approaches in Problem Solving

Problem-solving is a cognitive process that involves discovering, analyzing, and solving problems. There are various approaches to problem-solving. Choosing the appropriate problem-solving approach depends on the nature of the problem, available resources, and the specific goals of the problem-solving process. There are several common

approaches to problem-solving: Top-down approach, Bottom up approach, Trial and Error, Algorithmic Approach, Heuristic Approach, Divide and Conquer, Brainstorming etc.

Let us consider crafting a chair, and we have two approaches. The initial method involves starting with a holistic view, considering the chair's overall purpose, design aesthetics, and functionality. This is followed by breaking down the chair into essential components such as the seat, legs, backrest, and support structure. Each component is further analyzed, with sub- components like seat size, shape, material, and type of wood being considered. These elements are then systematically integrated to form the final chair.

On the other hand, the second approach entails initiating the chair-making process with a detailed design of each individual component. For instance, the seat is meticulously designed, considering factors like shape, size, and the type of wood to be used. Following the detailed design of each component independently, these components are subsequently joined together to create the unified final product.

Top-down and bottom-up are two complementary approaches in various fields, including software development, problem-solving, and system design. They refer to different methods of breaking down and addressing complex problems. The top-down approach goes from the general to the specific, and the bottom-up approach begins at the specific and moves to the general.

1.1.6.1 Top Down Approach

In a top-down approach, you start with an overview of the system or problem and then break it down into smaller, more manageable sub-systems or sub-problems. Fig 1.1.9 shows the design of top down approaches.

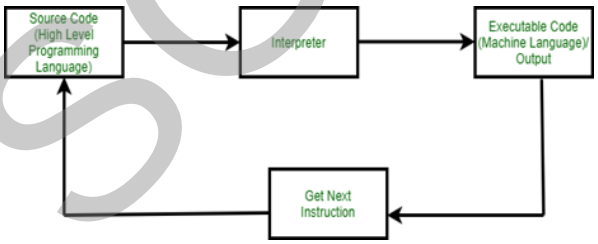


Fig 1.1.9 Top Down Approach

Process:

1. Begin with a broad understanding of the problem or system.
2. Decompose into Sub-Problems
3. Address Sub-Problems Sequentially

Advantages:

- Provides a clear overview of the entire system.
- Useful for planning and organizing complex tasks.
- Supports early identification of major components and their interactions.

Disadvantages:

- May overlook important details until the later stages.
- Dependencies between components may not be fully understood initially.

When to Use Top-Down Approach:

- When the overall structure of the system is critical
- In planning and organizing complex projects
- For projects where high-level design decisions are crucial

1.1.6.2 Bottom-Up Approach

In a bottom-up approach, you start with the details or smaller components and gradually build them up into a complete system or solution. Fig 1.1.10 shows the design of Bottom Up approach.

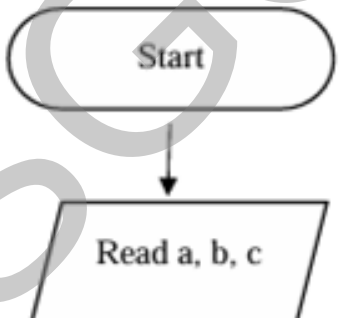


Fig 1.1.10 Bottom Up Approach

Process:

1. Begin with individual elements or details
2. Combine the smaller components into larger structures or systems
3. Build Upward

Advantages:

- Allows for early testing and validation of smaller components.
- Addresses details and specifics from the beginning.

- Incremental progress can be observed

Disadvantages:

- Initial lack of a clear overall structure.
- Integration challenges may arise as components are combined.

When to Use Bottom-up Approach:

- When the details and specifics are critical
- In situations where early testing of components is necessary
- For projects that allow for incremental development

1.1.7 Phases of Programming

A program is a set of instructions written by a programmer. Program contains detailed instructions and complete procedures for performing the relevant tasks.

Steps involved in Programming are given in Fig 1.1.11

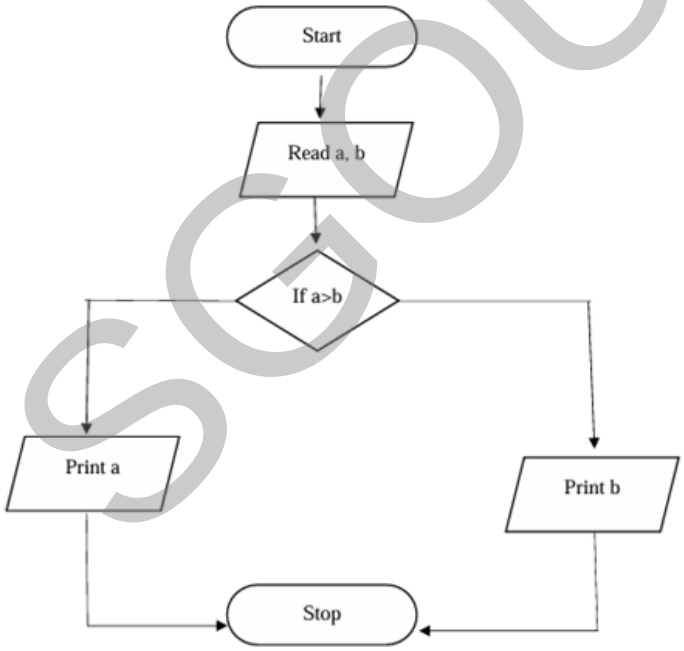


Fig 1.1.11 Programming Phases

1. **Problem Definition:** This is the initial phase, the problem or need is identified and defined. It involves understanding the requirements, constraints, and objectives of the project.
2. **Analyzing the Problem:** This involves understanding the needs of end-users, defining data structures, and specifying functionalities.
3. **Algorithm Design:** At this stage, the steps to be executed during different

phases of the program are defined. These steps are often described in simple English or pseudocode and can serve as a high-level strategy for implementation

4. **Flowchart:** It is a graphical tool that shows the steps/stages which are to be executed in a program. Making flowchart helps us in increasing our process of program development because it facilitates us to define the logic, detecting and removing errors in a program design.

5. **Coding:** The act of providing instructions using any computer language is referred to as coding. The entire coding process depends upon the information we obtained in previous steps. Choice of language depends upon the requirements and facilities available with a language.

6. **Debugging:** Debugging is the term used to describe the process of identifying and eliminating errors in a program.

7. **Testing:** In this stage we test the program by entering dummy data (includes usual, unusual and invalid data) to check the behavior and result of the program towards the given data.

8. **Final Output:** After going through all the above stages, the program is given the TRUE DATA. Here the programmer expects the positive results of the program and expects full efficiency of the program.

9. **Documentation:** It includes documenting requirements, design decisions, and coding details. There are two types of documentation

   - **User Manual** provides the user complete information about how to operate the program and what needs to be done when the user faces a problem.

   - **Technical Manual** contains the technical information about the program. This is used to get technical details of the program when the system is not working properly or requires modifications.

Recap

- A computer is a digital machine that takes input, processes it, stores it, and gives output.
- Data refers to raw facts, while information is processed, meaningful data.
- Data processing involves converting data into useful information.
- The CPU processes data and includes the ALU, control unit, and registers.
- Memory is divided into primary (RAM and ROM) and secondary (HDD, SSD, etc.).
- Machine and assembly languages are low-level, close to hardware.
- High-level languages like Python and Java are easier to use and understand.
- Compilers translate the entire code at once; interpreters do it line by line.
- Assemblers convert assembly code into machine language.
- An algorithm is a step-by-step solution to a problem.
- A flowchart visually represents an algorithm using standard symbols.
- Top-down approach breaks a problem into smaller parts from the top.
- Bottom-up approach builds individual parts first, then combines them.
- Programming phases include defining the problem, analyzing, designing an algorithm, creating a flowchart, coding, debugging, testing, and producing output.

Objective Type Questions

1. What is the temporary storage area in a computer called?
2. Which translator converts high-level code into machine language?
3. What symbol is used to indicate the start or end of a flowchart?
4. What is the main purpose of a flowchart?
5. What is the process of converting raw data into useful information?
6. What is the memory type that loses its content when the power is off?
7. What do we call the process of finding and fixing errors in a program?

8. What do we call the step-by-step instructions used to solve a problem?

9. What phase of programming involves defining the problem and understanding the requirements?

10. Which phase of programming involves testing the program with dummy data to check for errors and behavior?

Answers to Objective Type Questions

1. RAM
2. Compiler
3. Terminator
4. Visual representation of a process
5. Data processing
6. Volatile memory
7. Debugging
8. Algorithm
9. Problem Definition
10. Testing

Assignments

1. Explain different components of the computer system.
2. Distinguish between Primary and Secondary memory.
3. Explain algorithm with examples.
4. Draw flowchart to find largest among five numbers
5. Explain different phases of programming

References

1. Downey, A. B. (2015). *Think Python: How to think like a computer scientist* (2nd ed.). Green Tea Press

2. Dierbach, C. (n.d.). *Introduction to computer science using Python* (1st ed.). Wiley India Pvt. Ltd.

3. Chun, W. J. (2015). *Core Python applications programming* (3rd ed.). Pearson Education India.

4. Tamassia, R., Goldwasser, M. H., & Goodrich, M. T. (2016). *Data structures and algorithms in Python* (1st ed.). Wiley India Pvt. Ltd

5. Thareja, R. (2017). *Python programming using problem solving approach*. Oxford University Press.

Suggested Reading

1. Brookshear, J. G., & Brylow, D. (2019). *Computer science: An overview* (13th ed.). Pearson.

2. Schneider, G. M., & Gersting, J. L. (2018). *Invitation to computer science* (8th ed.). Cengage Learning

3. Forouzan, B. A. (2008). *Foundations of computer science* (2nd ed.). Cengage Learning.

4. Malik, D. S. (2009). *C++ programming: From problem analysis to program design* (5th ed.). Cengage Learning

5. Severance, C. R. (2017). *Python for everybody: Exploring data using Python 3* (1st ed.).

UNIT 2 Fundamentals of Python

Learning Outcomes

The learner will be able to:

- understand and apply the fundamental syntax and semantics of Python
- familiarisation of different data types
- familiarise different operators and their practical applications.
- demonstrate an understanding of control flow structures such as loops (for and while) and conditional statements (if, elif, else).

Prerequisites

The journey of language learning commences with the alphabet, progressing to words, sentences, and eventually, understanding the structural elements of a language. In the realm of programming languages, this progression is referred to as the character set and tokens.

Subject + Verb + Object

This is the basic structure of the English sentence. Similar to English, every language possesses its unique grammar or syntax. When dealing with programming languages, adherence to specific syntax and semantics is crucial, dictated by the language's structure. Understanding the language's flow becomes imperative. Only through mastering these aspects can we proficiently and effectively code programs tailored to various applications.

Keywords

Character set, Tokens, Keywords, Punctuators, Operator, Control statements

Discussion

Python is a powerful, versatile, and beginner-friendly programming language. Developed by Guido van Rossum and it was first released in 1991. It is a high level and general purpose programming language. Its readability and ease of use make it popular. Its applications range from web development and data science to artificial intelligence and scientific research. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. The language is popular and has plenty of libraries available, allowing programmers to get a lot done with relatively little code. Python is available for free and has an open-source license, allowing anyone to contribute to its development. Python is also a portable language that means its code can run on different operating systems without modification. It is generally considered to be among the top four or five most widely used programming languages in the world today. Python is often called a scripting language because it makes it easy to utilize and direct other software components.

Python, renowned for its emphasis on reducing development time, finds extensive usage across a diverse array of products and roles. Its user base includes prominent entities such as Google, YouTube, Industrial Light & Magic, ESRI, the BitTorrent file sharing system, NASA's Jet Propulsion Lab, the game Eve Online, and the National Weather Service. Python's versatility spans various domains, encompassing system administration, website development, mobile scripting, education, hardware testing, investment analysis, computer games, and spacecraft control.

1.2.1 Applications of Python

Python has a large number of applications; its use can be tailored to individuals based on their specific areas of interest and ideas.

- Desktop applications
- Automation and Scripting
- Education and teaching
- Web Development
- Artificial Intelligence
- Data Science and Machine Learning
- Game Development
- Network Programming

- Web Scraping
- Cybersecurity
- Internet of Things (IoT)
- Database Programming
- Financial and Trading Applications

1.2.2 Install Python in Our System

Depending on the operating system, the installation procedure of Python is different. Python is already installed with the Linux operating system. Python's infrastructure is intensively used by many Linux OS components.

Different steps to install Python

1. Visit the official website of Python https://www.python.org/downloads/
2. Click on the “Download Python 3.12.1” button (It is the latest version of Python, we can choose different version of Python)
3. Run the Installer
4. Make sure to select the "Add Python 3.x to PATH" checkbox so that you can easily run Python from the command line.
5. Follow the instructions in the installation wizard.
6. Click "Install Now" when prompted.
7. Open a command prompt (search for "cmd" in the Start menu).
8. Type python --version or python -V and press Enter.
9. You should see the Python version number, indicating a successful installation.

1.2.2.1 Python Online IDE (Integrated Development Environment)

To write a program using the Python programming language, we need an IDE. An IDE, or Integrated Development Environment, is a software application that provides a comprehensive set of tools for software development. There are several online Integrated Development Environments (IDEs) available for Python that allow you to write, run, and test Python code without installing anything locally on your computer. Few examples for online IDEs are PyCharm, Visual Studio Code, Spyder, Thonny, Repl.it, Google Colab, PythonAnywhere, Jupyter Notebooks on Microsoft Azure, IDEOne, PyFiddle.

In Google Colab, we can simply search for 'Google Colab' online. Once opened, it provides a notebook interface where we can write and run code line by line and view the output immediately.

1.2.2.2 Sample Hello world program- run and compile in online IDE

How to Start:

Let's start Jupyter Notebook Online as shown in fig 1.2.1 by opening the link

https://jupyter.org/try

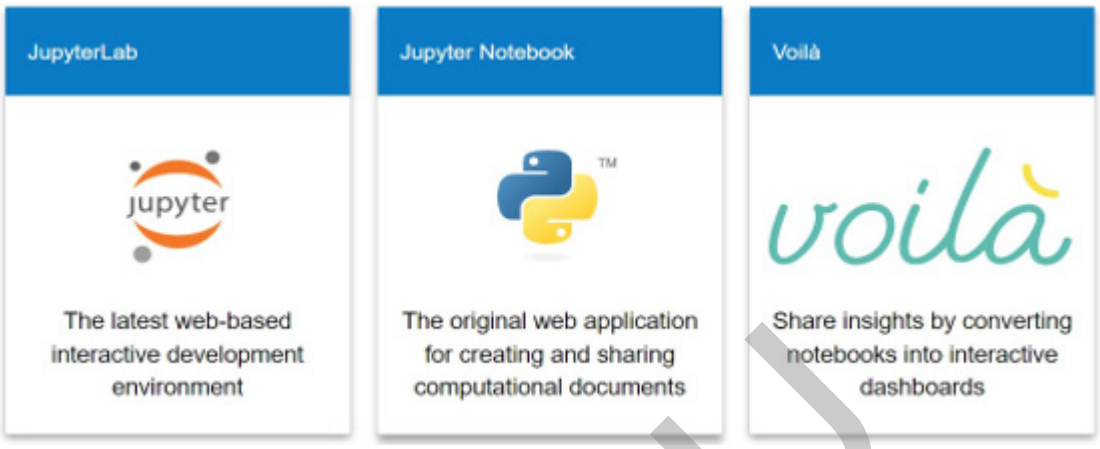


Fig 1.2.1 Python IDE

Start a new workbook as shown in fig 1.2.2. We can write the program in the cell provided by the IDE.

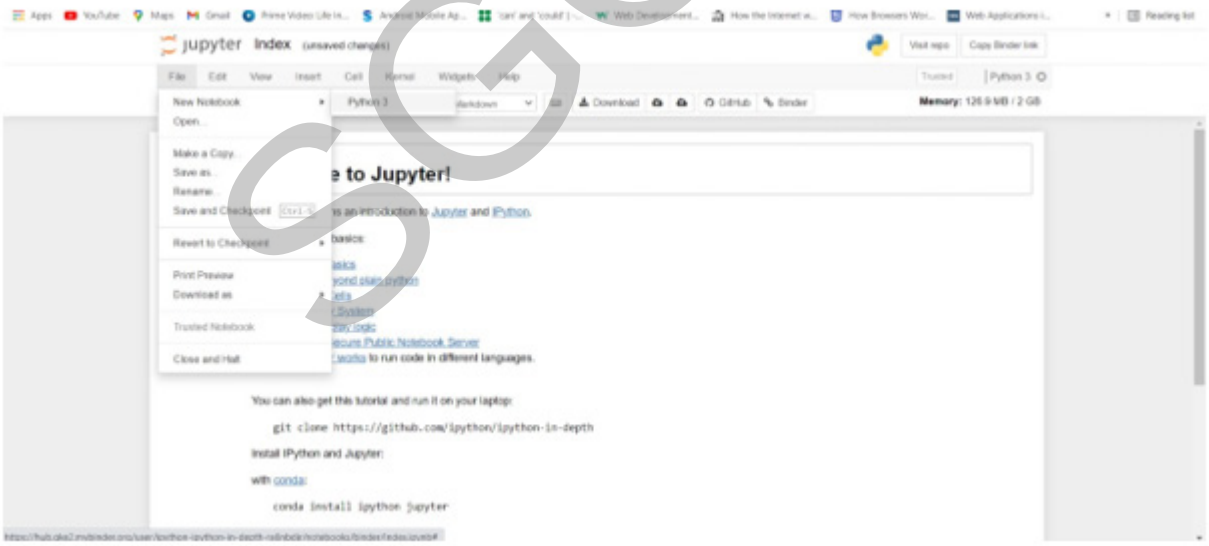


Fig 1.2.2 Python IDE

Fig 1.2.3 shows python IDE, here we can write programs

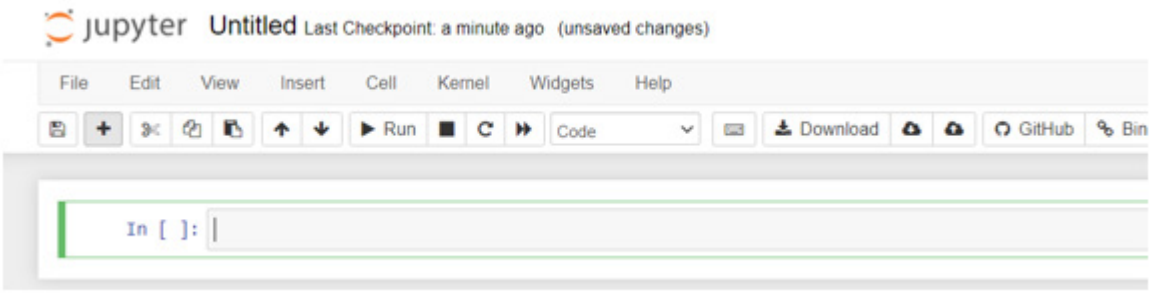


Fig 1.2.3 Python IDE

Type print ("Hello World") as shown in fig 1.2.4 below

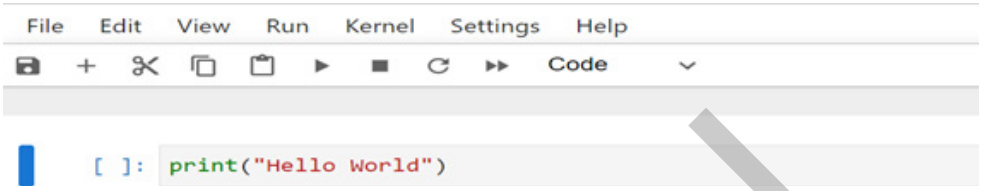


Fig 1.2.4 Python IDE

Click on the Run button and click on Run Selected Cells or click on ► as shown in fig 1.2.5 to execute the program and observe the result.

Fig 1.2.5 Python IDE

The following fig 1.2.6 shows the result will be displayed.

```
: print("Hello World")

Hello World
```

Fig 1.2.6 Python IDE

Congratulations, you have created a Python program and executed the same to see the result.

1.2.3 Character set

As we know, the study of any language begins. Similarly Python also has its own alphabet called character set. The set of valid characters in a language which are the fundamental units of that language is called character set. Python supports a vast range of characters, encompassing both ASCII and Unicode. This means you can work with a wide variety of languages and symbols within your code.

1. Letters

Both uppercase and lowercase alphabets are supported by Python Upper case letters A-Z

Lower case letters a-z

2. Digits

Digits 0-9

3. Special characters

Punctuation marks: , . ; : ! ? " ' ` ( ) [ ] { } + - * / = < > | & % ^ ~ # @ $ Other symbols: \ _ | \t (tab) \n (newline) \r (carriage return) \f (form feed)

4. White spaces

Space

Tab

Carriage

Return

Newline

Form feed

Unicode Characters

Python supports a vast array of Unicode characters, representing various languages and scripts. This includes accented characters, ideograms, emojis, and more.

1.2.4 Python Tokens

After learning the alphabet, in the Python character set, the second stage is learning words constituted by alphabets (characters). The word token is similar to words in natural languages. Tokens are the fundamental building blocks of the programming language. They are also known as lexical units. There five tokens in Python

1. Keywords
2. Identifiers

3. Literals
4. Punctuators
5. Operators

1.2.4.1 Keywords

Keywords are reserved words that convey special meaning to the interpreter. These are also known as reserved words as they are reserved by the language for special purposes and cannot be redefined for other purposes. The below table 1.2.1 shows different keywords.

Table 1.2.1 Some keywords in Python

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
elif	Used in conditional statements (else if)
else	Used in conditional statements
for	To create a for loop
if	To make a conditional statement
import	To import a module
in	To check membership in a list, tuple, etc.
is	To test if two variables refer to the same object
lambda	To create an anonymous function
not	A logical operator
or	A logical operator
return	To exit a function and return

1.2.4.2 Identifiers

Identifiers are user defined words that are used to name different program elements like variables, functions, classes, objects etc.

Rules for naming variable are :

1. The name must start with a letter or underscore.
2. Example : a, name, Name, Class1, _lastname, addFunction
3. The name cannot start with a number.
4. Names can contain letters(A-Z and a-z), digits(0-9), or underscores (_).
5. Python is case-sensitive. So names written in uppercase and lowercase are distinct (age, Age and AGE are three different variables).
6. A variable name cannot be any of the Python keywords.

1.2.4.3 Python Literals

We know the value of Pi(π) is 3.14. Similarly we can use a lot of constants in our program. In Python, literals are fixed values that represent constant data within your code. They are essential for assigning values to variables, performing operations, and building data structures. Different types of literals in Python:

1. Numeric Literals

- Integer Literals: Represent whole numbers without a decimal point. Examples: 10, -5, 0
- Floating-Point Literals: Represent numbers with decimal points. Examples: 3.14159, -0.001, 1.23e4 (scientific notation)
- Complex Literals: Represent complex numbers with a real and imaginary part. Examples: 5 + 3j, -2.1j

2. String Literals

Represent sequences of characters enclosed in single quotes ('), double quotes ("), or triple quotes (""""). Triple quotes allow for multi-line strings and preserving formatting.

Examples: 'Hello, world!', "Python is awesome!", '''This is a multi-line string.'''

3. Boolean Literals: Represent truth values: True and False.

Examples: is_valid = True, has_error = False

4. None Literal: Represents the absence of a value or a null value.

Example: result = None

5. Collection Literals: Represent groups of values.

6. List Literals: Enclosed in square brackets [ ], hold elements of any data type, ordered and mutable.

Example: numbers = [1, 4, 2, 5]

7. Tuple Literals: Enclosed in parentheses ( ), hold elements of any data type, ordered and immutable.

Example: coordinates = (3, 5)

8. Set Literals: Enclosed in curly braces { }, hold unique elements, unordered and mutable. Example: unique_letters = {'a', 'b', 'c'}

9. Dictionary Literals: Enclosed in curly braces { }, hold key-value pairs, unordered and mutable.

Example: person = {'name': 'Alice', 'age': 30}

Remember that literals provide a concise way to represent fixed values in your Python code, making it more readable and efficient.

1.2.4.4 Punctuators

For grammatical perfection of a sentence we use different punctuators like , ,:, !, () etc. that are symbols used to organize the code's structure and syntax. It is also called Delimiter.

Examples: (), [], {}, ,, :, ., ;, @, # (for comments)

1.2.5 Operators

Suppose we need to add 3, 2, then we use + operator. Operators are symbols that tell the interpreter to perform specific operations on variables and values. Variables on which operations are performed are called operands, and the operator applied to these operands is referred to as the opcode. Here addition of 3+2, 3 and 2 are operands and + is opcode.

Different operators are

- Arithmetic operators : +, -, *, ?, %
- Logical operators : ‘and’, ‘or’,‘not’
- Relational operators : <, <=, >, >=, ==, !=
- Bitwise operators: &, |, ^, ~, <<, >>
- Assignment operators: =, +=, -=, *=, /=, etc.

1.2.6 Variables

We utilize clock rooms to securely store our bags. Each storage unit is assigned a unique name or number, which we use to identify and locate our bag within the facility. In programming languages, this identifying number or name is referred to as a variable. Different variables are stored in different memory locations.

A variable is a named memory location used to store values

This data can be of various types, such as numbers, text, or complex objects, and it can change during the execution of a program. Variables allow programmers to work with and manipulate data within their programs. They act as placeholders for values that can be used or modified throughout the course of a program's execution. In Python, there is no need to declare variables separately. In Python, dynamic typing allows you to assign values to variables without explicitly declaring their types. However, if you want to check the type of a variable at runtime, you can use the type() function.

Variable Initialization or Assignment

Variables are assigned values using the assignment operator (=). The value can be changed as needed.

Age = 12 , 'Age' is a variable storing the value 12

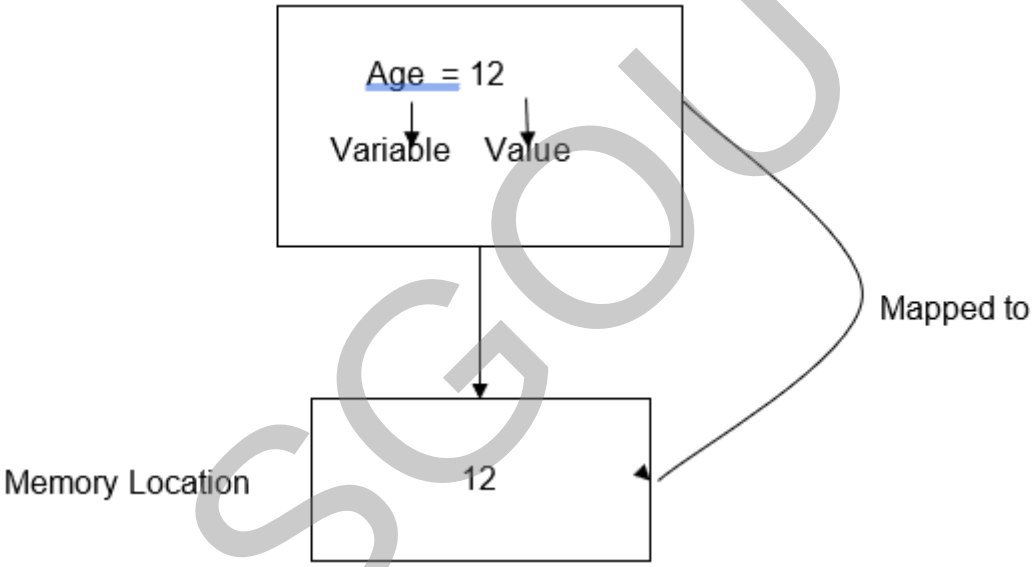


Fig 1.2.7 Memory allocation of a variable

Age = 30 # Value updated to 30

Variables are of different data types, such as integers, floating-point numbers, strings, etc. The data type determines the kind of values a variable can hold.

name = "John" # 'name' is a string variable

salary = 50000.75 # 'salary' is a floating-point variable

Dynamic Typing: Some programming languages, like Python, support dynamic typing, allowing a variable to change its data type during runtime.

x = 5 # 'x' is an integer

x = "Hello" # 'x' is now a string

Scope of a Variable

Variables have a scope, which defines the region of the program where the variable is accessible. There are two types of variables based on scope,

1. **Local variables:** These are limited to a specific block or function where it is initialized.

2. **Global variables:** That can be accessed throughout the entire program.

Naming rules of variables

1. Python is case sensitive, which means Age and age are two different variable names.
2. The first character of the variable name must be an alphabet or an underscore (_).
3. The rest of the variable name can consist of letters, underscores, or digits.
4. Special symbols (*,#,& etc) and space are not allowed for variable names.
5. Python keywords are not allowed as variable names.

1.2.7 Data Types

Data is processed by applications or programs. For example, in the student registration process, we are using different data such as name, date of birth, address, family monthly income, etc. Different types of data are used in the registration process. Date of birth is date type, monthly income is numeric data, a name is a group of letters.

A data type is a classification of data. Data type is a type of data or variable which is used in programs . Memory location will be allocated according to the type of data. For example, the memory requirement of storing the values “KKG”,5, and 5.10 will be different.

In Python, we need not declare a datatype. In Python, dynamic typing allows you to assign values to variables without explicitly declaring their types. However, if you want to check the type of a variable at runtime, you can use the type() function.

Example :

msg = “Hello, Python” print(type(msg))

Output: <class ‘str’>

Figure 1.2.8 shows different data types of Python. Python supports different data types as detailed below:

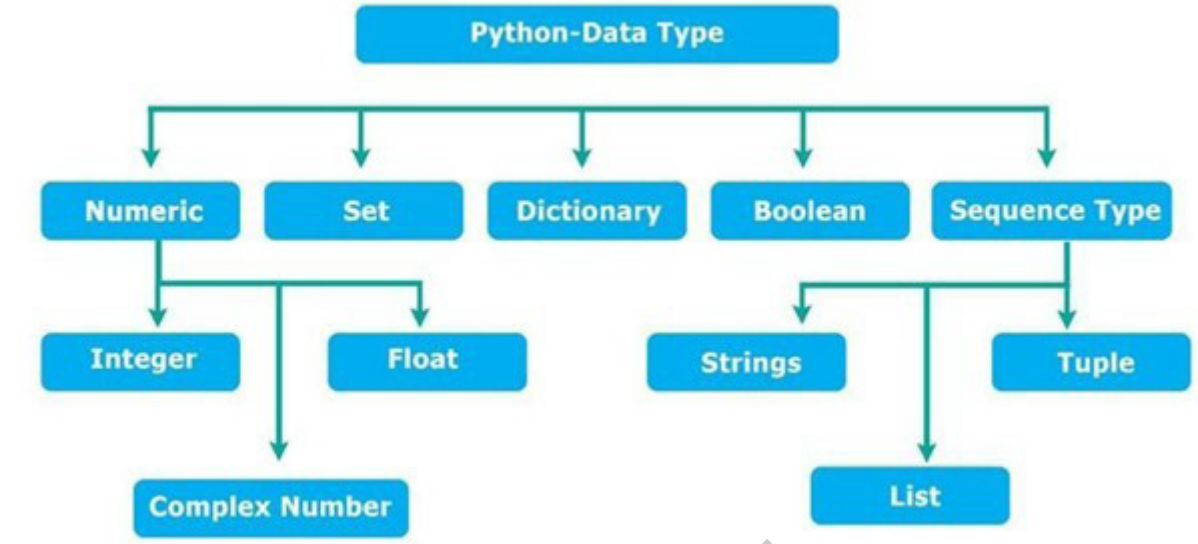


Fig 1.2.8 Data types in Python.

1.2.7.1 Numeric Data Types

1. **int :** It represents integer numbers(whole numbers). Integers can be Octal and Hexadecimal also.
2. Example, x = 7, a= 90.
3. **long : It** holds long integers.
4. **float :** Represents a decimal number (Decimal numbers). Example: Price = 20.5 , Pi = 3.14
5. **Complex numbers:** It represents complex numbers, numbers with real parts and imaginary parts. Complex numbers are used in geometry, scientific calculations, and calculus.

Example: 3+4i, here 3 is the real number and 4i is the imaginary number. The value of i = √-1.

1.2.7.2 Sequence Data Types

1. String data type

A string in Python is a sequence of characters. Characters are letters, numbers, symbols, etc. Strings are used when you need to process text data like names, addresses, etc. Data in between the quotes “ ” are string data.

Example: “Hello World”, “Keralam”

“222345” is not a number, it is a string. Guess the reason?

Different types of string representation

a. Single quotes string : Strings can be defined using single quotes. Single quotes are useful when you want to include double quotes within the string without escaping them.

Example:

my_string = 'Hello, Python!'

my_string = 'He said, "Hello!"'

b. Double quotes string : Strings can also be defined using double quotes. Double quotes are useful when you want to include single quotes within the string without escaping them.

Example:

my_string = "Hello, Python!"

my_string = "She's a programmer"

c. Triple quotes for multiline strings : Three single quotes or double quotes can be used (''' or """). Triple quotes are used for multiline strings. They can span multiple lines without the need for escape characters.

Example :

message = '''Programming is fun.

Python is a high-level language.

Python is used by Facebook, Google, and other companies."'

When we input the data from the keyboard, the number will be considered as string only. See the following example and output.

mark = input("Enter a mark: ") # input() function is used to take user input from the keyboard

print(mark)

print(type(mark))

Output

Enter a mark: 67.9 67.9

<class 'str'>

To read a number as a float number, Python uses Typecasting to convert one data type to another.

Example

mark = float(input("mark")) print(mark) print(type(mark))

Input the mark 67.9 and the output will be 67.9

<class 'float'>

The second method of data type casting is Mark = float(mark)

Example

```
a =int(input("enter first number"))

b =int(input("enter second number"))

c =float( a) + float(b) print(c)
```

List type

The list can be used to store multiple items or values or data in a single variable name. Ordered, mutable collection of items.

For example name = [“KKG”, “pen”,”beach”]. ‘name’ represents a list. If we write name = “KKG”, name is a string variable name that stores only one data.

Tuple

Tuples stores multiple items under the single variable name. Ordered, immutable collection of items. Tuple items are unchangeable, ordered, and allow duplicate values. Tuples will be used when the list remains unchanged. If you wanted to add a new car to a list of cars, the Python list would be suitable.

```
cars=( "Toyota","KIA", "Maruthi")  print(cars)
```

Toyota, KIA,Maruthi

Range

Represents a sequence of numbers, often used in for loops, e.g., range(5) generates 0, 1, 2, 3, 4.

1.2.7.3 Boolean data type

bool represents logical values that are ‘True’ or ‘False’. Example: flag = ‘true’

1.2.7.4 Set Data Type

set : A set is a data collection type used for storing multiple items in a single variable. Sets are unordered and are not always consistent in the order they get returned. Items in the set are immutable ie. cannot be changed is called frozenset. However, items can be added and removed. Duplicates are not permitted and will be removed. It is also noteworthy that the Python set supports any data type including numbers, strings, and booleans but also has some support for nested data structures.

Eg:- myset = {1, 2, "apple"}

1.2.7.5 Dictionary type

dict : Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered, changeable and does not allow duplicates.

e.g., my_dict = {'name': 'Appu', 'age': 14, 'class': '9th'}

1.2.8 Operators in Python

When creating an ATM program setup, we utilize the addition operation (+) when customers make deposits and the subtraction operation (-) when they withdraw funds. These operations involve the use of operators. Operators are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations.

> Variables on which operations are performed are called operands, and the **operator** applied to these operands is referred to as the **opcode.**

Here addition of 3+2, 3 and 2 are operands and '+' is opcode.

Types of operators in Python

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Identity Operators and Membership Operators

1.2.8.1 Arithmetic Operators

Python Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, and division.

Table 1.2.2 Different arithmetic operators

Operator	Syntax	Description
+	x+y	Addition: adds two numbers
-	x-y	Subtraction : subtracts two numbers
*	x*y	Multiplication : Multiplies two numbers. It is also used to replicate strings.
/	x/y	Division (float): divides the first operand by the second.
//	x//y	Division (floor): divides the first operand by the second and returns the integer value.

%	x%y	Modulus: returns the remainder when the first operand is divided by the second.
**	x**y	Power: Returns first raised to power second.

1.2.8.2 Comparison Operators

Comparison operators are used to compare two values and return the result as boolean value 'True' or 'False'. It is also called relational operators.

Table 1.2.3 Different comparison operators

Operator	Syntax	Description
==	x==y	Equal to, it return 'True' if x equal to y, otherwise 'False'
<	x<y	Less than, it return 'True' if x less than y, otherwise 'False'
<=	x<=y	Less than or equal to, it return 'True' if x less than or equal to y, otherwise 'False'
>	x>y	Greater than, it return 'True' if x greater than y, otherwise 'False'
>=	x>=y	Greater than, it return 'True' if x greater than or equal to y, otherwise 'False'
!=	x!=y	Not equal to, it return 'True' if x not equal to y, otherwise 'False'

1.2.8.3 Logical Operators

Logical operators are used to combine conditional statements and return boolean value 'True' if both statements are true otherwise false.

Table 1.2.4 Different logical operators

Operator	Syntax	Description
and	x>1 and x<10	Return true if both statements are 'True', that is x>1 and x<10, otherwise return 'False'
or	x<10 or x>5	Return true if any one of the condition is true that is x<10 or x>5 otherwise return false
not	not(x<10 and x>5) not(x<10 or x>5)	Reverse the result, returns False if the result is true

1.2.8.4 Bitwise Operators

Table 1.2.5 Different bitwise operators

operator	Syntax	Description
& (AND)	x&y	The & operator compares each bit and set it to 1 if both are 1, otherwise it is set to 0 Example x=6 and y=3, x= 0000000000000110 y= 0000000000000011 x&y= 0000000000000010 , that is x&y = 2
| (OR)	x|y	The ^ operator compares each bit and set it to 1 if only one is 1, otherwise (if both are 1 or both are 0) it is set to 0 Example x=6 and y=3 x= 0000000000000110 y= 0000000000000011 x|y= 0000000000000111 , that is x|y = 7
^(XOR)	x^y	The ^ operator compares each bit and set it to 1 if only one is 1, otherwise (if both are 1 or both are 0) it is set to 0 Example x=6 and y=3 x= 0000000000000110 y= 0000000000000011 x^y= 0000000000000101 , that is x^y = 5
~(NOT)	~x	The ~ operator inverts each bit (0 becomes 1 and 1 becomes 0) Example x=3 x= 0000000000000011 ~x= 1111111111111100 , that ~x=-4
<<(Zero fill left shift)	x<<value	The << operator inserts the specified number of 0's (in this case 2) from the right and let the same amount of leftmost bits fall off Example x=3 x= 0000000000000011 x<< 2= 0000000000001100 , that is push 00 in from the left to right then x<<2 = 12
>>(signed right shift)	x>>value	The >> operator moves each bit the specified number of times to the right. Empty holes at the left are filled with 0's. Example x=8 x= 0000000000001000 x>>2= 0000000000000010 , move each bit 2 times to the right then x>>2 = 2

1.2.8.5 Assignment Operators

Assignment operators are used to assign values to variables

Table 1.2.6 Different assignment operators

Operator	Syntax	Description
=	x=5	Assign value 5 to the variable x
+=	x+=5	Equal to x=x+5, add the value of x with 5 and store the result into x
-=	x-=5	Equal to x=x-5, subtract the value of x with 5 and store the result into x
*=	x*+5	Equal to x=x*5, multiply the value of x with 5 and store the result into x.
/=	x/=5	Equal to x=x/5, divide the value of x with 5 and store the result into x.
%=	x%=5	Equal to x=x%5, find x mod 5 and store the result into x
**=	x**=5	Equal to x=x**5, find x raise to 5 and store the result into x
&=	x&=5	Equal to x=x&5, find x & 5 and store the result into x this comparison returns False because x is equal to y

1.2.8.6 Membership operators

Membership operators are used to test if a sequence is presented in an object.

Table 1.2.7 Different membership operators

Operator	Syntax	Description
in	x in y	Returns True if a sequence with the specified value is present in the object. x = ["apple", "banana"] print("banana" in x) # returns True because a sequence with the value "banana" is in the list

not in	x not in y	Returns True if a sequence with the specified value is not present in the object. x = ["apple", "banana"] print("pineapple" not in x) # returns True because a sequence with the value "pineapple" is not in the list

1.2.9 Statements in Python

In Python, there are various types of statements used to perform different actions in a program. Statements are instructions that Python executes to perform actions. They form the building blocks of Python programs.

Different statements are:

1. Assignment statements

Used to assign values to variables.

Eg:- x=2, name=”xyz”

2. Expression statements

Used to evaluate expressions. An expression can be a combination of values, variables, operators, and function calls.

Eg:- sum=3+4, string=”french”+”fries”

3. Conditional statements

Used to execute different blocks of code based on conditions.

Eg:- if(x>y)

print (x)

4. Looping statements

Used to execute a block of code repeatedly. for i in range(1,10)

print i

5. Return statements

Used to return a value from a function. def add(x, y):

return x + y

6. Pass statements

Used as a placeholder for future code or to create empty code blocks.

if x < 5:

pass # Placeholder for future code

1.2.10 Control structures

Control structures define the flow of a program's execution. Based on the results of relational or logical operations, the program's flow may deviate. These structures enable you to make decisions, repeat actions, and control the order in which code blocks are executed. Various control structures exist, including:

1. Conditional statements
2. Loop statement
3. Loop control statements

1.2.10.1 Conditional statements (Decision making statements)

Conditional statements enable a program to evaluate expressions and execute code only if certain conditions are true or false. Decision-making is a particular situation in a programming language in which we need to make some decisions. Based on these decisions we will execute the next block of code. It decides the direction of the flow of program execution.

In python, there are four types of decision making statements.

- if statements
- if else statements
- nested if statements
- if-elif ladder statements

1. if statements

If statement is the simplest decision-making statement. It decides whether certain statements need to be executed or not. It contains a body of statements that runs only when the condition given in the if statement is true. If the condition is false, then the statements are skipped over and not executed.

Syntax:

if expression: statement (s)

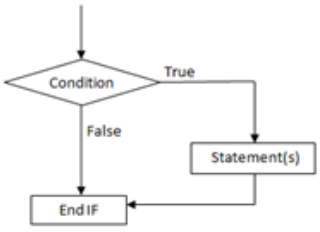


Fig. 1.2.9 Flow chart of if statement

Here the Condition represents a condition which is either a relational expression or logical expression. Remember to use indentation to define the blocks of code under each condition. The colon (:) is used to indicate the beginning of a new block of code

for the if statement.

Example : Write a program to print a given number if it is greater than 10.

```
num=89
if num>10:
        print(num, "greater than 10")
print(“Excellent")
```

Output:

89 greater than 10 Excellent

2. if else statements

In if else statement if the condition of if statement is true, then all the statements which are

written under if the statement will execute, otherwise the else part will execute. The else block should be right after the if block and it is executed when the expression is False.

Syntax:

```
if expression:
statement(s)
else:
statement(s)
```

Example: Write a program to find largest among two numbers

```
x= int(input("Enter an integer as input: ")) y= int(input("Enter an integer as input: "))
if x>y:
        print(x, "is larger than", y, "among", x,y)
else:
        print(y, "is larger than", x, "among", x,y)
```

Output:

Enter an integer as input: 3 Enter an integer as input: 67 67 is larger than 3 among 3 67

3. Nested if statements

A nested if else statement means an if statement or if-else statement present inside another if or if-else statement. This can be implemented in three ways. In the first method, if statement can be placed inside the if code block and in second method, if statement can be placed inside the else statement and in third method in both if and else statement.

Syntax:

First Method

```
if expression:
        if expression:
                statement(s)
                        else:
                        statement(s)
                else:
        statement(s)
```

Second Method

```
if expression:
        statement(s)
else:
        if expression:
                statement(s)
        else:
                statement(s)
```

Third method

```
if expression:
        if expression:
                statement(s)t
        else:
                statement(s)
else:
        if expression:
                statement(s)
        else:
                statement(s)
```

Example:

Write a program to find largest among three numbers

```
x = int(input("Enter an integer as input: "))
```

```
y = int(input("Enter an integer as input: "))
z = int(input("Enter an integer as input: "))
if x > y:
    if x > z:
        print(x, "is the largest among", x, y, z)
    else:
        print(z, "is the largest among", x, y, z)
else:
    if y > z:
        print(y, "is the largest among", x, y, z)
    else:
        print(z, "is the largest among", x, y, z)
```

Output:

```
Enter an integer as input: 7
Enter an integer as input: 2
Enter an integer as input: 9
9 is largest among 7 2 9
```

4. if -elif -else ladder

In Python, there is no explicit "else if" syntax. Instead, you use the "elif" keyword to create an "if-elif-else" ladder. This construct allows you to create a sequence of conditions and execute the block of code associated with the first true condition.

Syntax:

```
if expression:
        statement(s)
elif expression:
        statement(s)
elif expression:
        statement(s)
else:
        statement(s)
```

In this context, the conditions are examined sequentially. If the first "if" condition is

true, its corresponding block is executed, and the rest of the ladder is skipped. If the first "if" condition is false, the program moves to the first "elif" condition. If the "elif" condition is true, its block is executed, and the remaining conditions are skipped. If none of the conditions is true, the "else" block is executed.

Example :

Write a program to perform arithmetic operations based on choice x=

int(input("Enter an integer as input: "))

y= int(input("Enter an integer as input: ")) op=

input("Choose an operator +,-,*,/,//,% :") if

op=="+":

print("Sum of", x, " and",y,"= ", x+y) elif

op=="-":

print("Minus of", x, " and",y,"= ", x-y) elif

op=="*":

print("Multiplication of", x, " and",y,"= ", x*y) elif

op=="/":

print("Division of", x, " and",y,"= ", x/y) elif

op=="//":

print("Integer division of", x, " and",y,"= ", x//y)

else:

print(" Mode of ", x, " and",y,"= ", x%y)

Output:

Enter an integer as input: 8

Enter an integer as input: 5

Choose an operator +,-,*,/,//,% ://

Integer division of 8 and 5 = 1

1.2.10.2 Loop Statements (Iteration statements)

I would like to create a beaded chain with a consistent color and pattern. To begin, I gather beads of the same color, ensuring I have enough for the desired length; in this case, I'll need 50 beads. Next, I select a thread to string the beads onto. Starting with one bead, I thread it onto the string, followed by the second, and so on. This process is repeated 50 times until all beads are strung onto the thread. Finally, I tie a secure knot at each end to prevent the beads from slipping out.

Initially collect 50 beads Then select thread

Repeat the following steps in 50 times:

Take one bead

Thread it onto the string

After that, put a tie on both ends.

> A loop is used to execute a statement or a group of statements several times. It iterates over a set of code a specified number of times.

We use a counter to know how many times the process is executed. The value of this counter decides whether to continue the execution or not. Since loops work on the basis of such conditions, a variable like the counter will be used to construct a loop. This variable is generally known as loop control variable because it actually controls the execution of the loop. Every loop has four parts:

1. **Initialisation:** Before entering a loop, its control variable must be initialized. During initialisation, the loop control variable gets its first value. The initialisation statement is executed only once, at the beginning of the loop.

2. **Test expression:** It is a relational or logical expression whose value is either True or False. It decides whether the loop-body will be executed or not. If the test expression evaluates to True, the loop-body gets executed, otherwise it will not be executed.

3. **Update statement:** The update statement modifies the loop control variable by changing its value. The update statement is executed before the next iteration.

4. **Body of the loop:** The statements that need to be executed repeatedly constitute the body of the loop. It may be a simple statement or a compound statement.

Python supports three types of looping statements.

- for loop
- while loop
- nested loops

1. for loop

The for loop is used to iterate over a sequence (such as a list, tuple, string, or range) or other iterable objects.

Syntax:

for var in sequence:

statement(s)

- for : Keyword that initiates the loop.
- var : Variable that holds the current item in each iteration.
- in: Keyword that links the item variable to the iterable.
- Sequence (iterable) : A sequence (like a list, tuple, string, etc.) or any object that can be iterated over.
- # code to be executed...: The code block that runs for each item in the iterable.

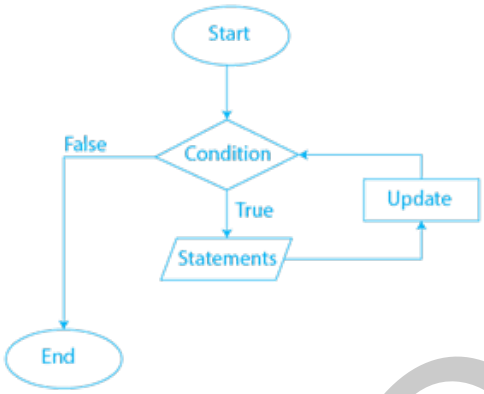


Fig 1.2.10 structure of for loop

Example 1: Write a program to print name of fruits in the given list

```
fruits = ["apple", "banana", "cherry"]
    for fruit in fruits:
        print(fruit)
```

Output:

apple

banana

cherry

Example 2: Write a program to print first 10 natural numbers n=11

```
for i in range(1,n):
    print(i)
```

2. while loop

The while loop continues to execute a block of code as long as a specified condition is True. The while loop is characterized as an entry-controlled loop because it first checks the condition before entering the loop. If the condition evaluates to True, only then will the body of the loop be executed. After the first iteration, the test expression is again

checked. This process will continue until the test expression becomes False.

Syntax:

```
while expression:
        statement(s)
```

Example : Write a program to find sum of even numbers up to 50

```
sum = 0
ev = 2
while ev <= 50:
   sum += ev
   ev += 2
print("Sum of even numbers up to 50:", sum)
```

Output

sum of even numbers up to 50 : 650

3. Nested loops

A nested loop means loops inside the loops. The inner or outer loop can be any type, such as a while loop or for loop. For example, the outer while loop can contain an inner for loop and vice versa. There is no limitation on the chaining of loops. The inner loop will execute one time for each iteration of the outer loop. The number of iterations will be equal to the number of iterations in the outer loop multiplied by the iterations in the inner loop.

Syntax for a nested for loop within for loop

```
for iterating_var in sequence:
        for iterating_var in sequence:
                statement(s)
```

Syntax for a nested while loop within while loop

```
while expression:
        statement(s)
```

Example: Write a program to print triangle of stars

```
height=int(input("Enter height of triangle : "))
for i in range(0,height+2):
        print("\t")
```

```
    for j in range(1,i+1):
        print("*", end="")
```

Output

```
Enter height of triangle : 5
*
**
***
****
*****
```

1.2.10.3 Loop control statements

In Python, loop control statements are used to change the normal flow of loop execution. These can be used if you wish to skip an iteration or stop the execution. There are three main loop control statements: break, continue, and pass

1. break

The break statement is used to exit the loop prematurely. When the break statement is encountered inside a loop, the loop is terminated immediately, and the program continues with the next statement after the loop.

Syntax :break

Example :

```
for i in range(0,7):
    print(i) if(i==5):
        break
print("numbers up to five")
```

2. continue

The continue statement is used to skip the rest of the code inside the loop for the current iteration and move to the next iteration.

Syntax :continue

Example:

```
for i in range(10):
if i % 2 == 0:
        continue # Skips even numbers
    print(i)
```

3. pass

The pass statement is used as a placeholder when a statement is syntactically required but you don't want any code execution. It does nothing.

Syntax :pass

```
for i in range(10):
        if some_condition:
                pass # Does nothing in this case else:
                print(i)
```

1.2.11 Functions

Modular programing

Let us consider the case of a school management software. It is a very large and complex software which may contain many programs for different tasks like admission, evaluation, accounts, staff management etc. The complex task of school management can be divided into smaller tasks or modules and developed in parallel, and later integrated to build the complete software.

In programming, the entire problem will be divided into small sub problems that can be solved by writing separate programs. This kind of approach is known as modular programming. Each sub task will be considered a module and we write programs for each module. The process of breaking large programs into smaller subprograms is called modularisation. Computer programming languages have different methods to implement modularization. The sub programs are generally called functions.

Advantages of modular programming

- Reduce the size of the program
- Less chances of errors
- Reduce programming complexity
- Improves reusability

Function

Let us consider the case of a coffee making machine and discuss its functioning. Water, milk, sugar and coffee powder are supplied to the machine. The machine processes it according to a set of predefined instructions stored in it and returns the coffee which is collected in a cup. The instruction-set may be as follows:

1. Get 60 ml milk, 120 ml water, 5 gm coffee powder and 20 gm sugar from the storage of the machine.
2. Boil the mixture
3. Pass it to the outlet

Usually there will be a button in the machine to invoke this procedure. Let us name the button with the word coffeemaker. Symbolically we can represent the invocation as:

Cup = coffeemaker (Water, Milk, Sugar, Coffee Powder)

We can compare all these aspects with functions in programs. The word coffeemaker is the name of the function, water, milk, sugar and coffee powder are parameters for the function and coffee is the result returned. It is stored in a cup. Instead of a cup, we can use a glass, tumbler or any other container.

> Function is a named unit of statements in a program to perform a specific task as part of the solution

A function is a block of statements that performs a particular task. The main idea behind the function is to put some commonly or repeatedly done tasks together and can use functions whenever we need to perform the same task multiple times without writing the same code again. As our program grows larger and larger, functions make it more organized and manageable.

Types of function

There are two types of functions:

- user defined functions
- built-in functions

1.2.11.1 User defined functions

User-defined functions are those functions that we define ourselves to do certain specific tasks. It enables you to group a set of instructions together under a unique name, enhancing the modularity and organization of your code. This named entity becomes a custom function within your program, allowing you to call it repeatedly with different inputs.

a. Creating a function

We can define our own functions in Python by using the "def" keyword.

The syntax of a Python function is

```
def function_name(parameters):
"""docstring"""
# function body
return [expression] # optional return statement
```

- **def keyword** is used to define a function.

- **Function name :** Any name can be given to function. Normally function name is related to the behavior of function.
- **Parameter (argument) :** Is the value of the variable passed to the function. Parameter is optional, and you can specify any number of parameters or none at all.
- **"""docstring"""** is an optional documentation string that describes what the function does. It's recommended to include it for clarity.
- **Function body :** contains the code that the function executes. This can include variable declarations, conditional statements, loops, etc.
- **return :** It's used to return a value from the function. Return is an optional statement, if no return statement is encountered, Python returns None by default.

To define a function in Python, we use the **"def"** keyword, followed by the **function name,** which should be a valid identifier in Python. If the function takes any parameters, they are placed within parentheses and separated by commas.

A docstring, which is an optional multi-line string enclosed in triple quotes (""""), can be included right after the function definition. This docstring provides a brief description of the function's purpose, parameters, and return values.

The function body comprises the code block that specifies the behavior of the function. It starts with a colon (:) and is indented consistently. The body can contain multiple statements, all indented at the same level.

If the function is expected to return a value, the "return" statement is used to specify the value(s) to be returned. The return statement is optional, and if it is not included, the function will automatically return None. It is possible to return a single value or multiple values separated by commas.

Here is a simple python function definition:

```
def greet(name):
    """Prints a greeting message."""
    print("Hai, " + name + "!")
```

In this example, we have defined a function called greet that takes a parameter called name. The function's purpose is to print a greeting message to the console. When the function is called with a specific name, it will print "Hai, " followed by the provided name and an exclamation mark.

b. Calling a Function

To call a function in Python, you simply write the function name followed by parentheses. If the function requires any arguments, you provide them inside the parentheses.

To call the above example function, we write: greet("Ann")

Here the passed string argument is "Ann". The function is then executed, resulting in the output "Hai, Aan!" being displayed on the console.

Arguments

Arguments are the values that you pass to a function during its invocation. They serve as inputs for the function to perform specific operations or calculations. Python supports various types of arguments, including:

1. Positional Arguments: These arguments are provided to a function in the same order as they are defined in the function's parameter list. The values are assigned to the respective parameters based on their positions.

Example:

def add_numbers(x, y): """Adds

two numbers.""" return x + y

result = add_numbers(3, 5)

print(result) #

Output: 8

In this example, 3 is assigned to x and 5 is assigned to y based on their positions.

2. Keyword Arguments: With keyword arguments, you explicitly specify the parameter name followed by the corresponding value, separated by an equal sign. This allows you to pass arguments in any order, disregarding their position in the parameter list.

Example:

def add_numbers(x, y):

"""Adds two numbers."""

return x + y

result = add_numbers(y=4, x=2)

print(result)

Output: 6

Here, the function is called with keyword arguments, allowing us to specify the values explicitly.

3. Default Arguments: Default arguments have predefined values assigned to them in the function's parameter list. If an argument is not supplied during the function call, the default value is used instead.

Example:

def greet(name, message="Hai"): """Prints

a personalized greeting."""" print(message

+ ", " + name) greet("Ann") # Output: Hai,

Ann greet("Balu", "Hello")

Output: Hello, Balu

In this example, the message parameter has a default value of "Hai". If not provided, the default value is used.

4. Variable-length Arguments: Python functions can accept a varying number of arguments. To achieve this, you can use the asterisk (*) before a parameter name for variable-length positional arguments, or two asterisks (**) for variable-length keyword arguments.

Example:

def add(*numbers):

"""Addition of numbers."""

total = sum(numbers) return

total

result = add(1, 2, 3, 4, 5) print(result)

Output: 15

The function add accepts a variable number of positional arguments using *numbers.

The arguments are treated as a tuple inside the function.

Return Values

Return values in Python pertain to the values that a function can provide to the caller once it has executed its tasks. The return statement is employed to indicate the specific value that a function will return.

1. Single Value Return: To return a single value, a function employs the return statement followed by the value that will be returned. This allows the function to provide a single result to the caller.

Example:

```
def multiply(a, b):
return a * b
result = multiply(3, 4)
print(result)
```

Output: 12

By utilizing the return statement, the multiply function returns the product of two numbers, and the resulting value is assigned to the variable "result."

2. Multiple Value Return: Functions have the ability to return multiple values by listing them separated by commas within the return statement. This allows the function to provide multiple results as a tuple or any other sequence type.

3. Empty Return: In situations where a return statement is encountered without a value or if a function lacks a return statement, Python implicitly returns None.

Example:

```
def is_even(number):

if number % 2 == 0:

        return True

else:

        return

result1 = is_even(4) result2 = is_even(5)

print(result1)

# Output: True print(result2)

# Output: None
```

In this example, we have a function called is_even() that checks whether a given number is even. If the number is divisible by 2, the function returns True using the return True statement. If the number is not even, the function does not explicitly provide a return value.

When we call is_even() with the number 4, the function returns True, indicating that 4 is an even number. We assign the return value to the variable result1 and print it, which outputs True.

When we call is_even() with the number 5, which is an odd number, the function does not have a return statement for this case. In such situations, Python implicitly returns None. We assign the return value to the variable result2 and print it, which outputs None.

1.2.11.2 Built-in Functions

Built-in functions are an integral part of Python's standard library, offering a broad range of functionalities that streamline coding tasks. These functions are readily accessible without the need for additional installation or setup. An example of such a fundamental built- in function is print(), which enables us to display text or variables on the console. By simply passing the desired content as arguments, we can swiftly output information to the user. For instance, executing print("Hello, world!") will print the phrase "Hello, world!" on the console.

Python's built-in functions provide numerous ways to manipulate and analyze data. One such function is sorted(), which returns a new list containing the sorted elements from the input iterable. For instance, using sorted([5, 2, 7, 1, 3]) will yield [1, 2, 3, 5, 7], showcasing how the function arranges the elements in ascending order. Another useful built-in function is len(), which determines the length of an object, such as a string, list, or tuple. By employing len(), we can quickly ascertain the number of elements in a given collection. For example, len("Python") will return the value 6, representing the length of the string "Python".

Python's built-in functions also facilitate convenient mathematical operations. The abs() function, for instance, returns the absolute value of a number, disregarding its sign and providing the positive value. Hence, executing abs(-5) will yield 5. Additionally, the round() function allows us to round a floating-point number to a specified number of decimal places. For example, round(3.14159, 2) will return 3.14, rounding the number to two decimal places. These built-in mathematical functions offer flexibility when performing calculations and are commonly employed in various applications.

Recap

- Python is a popular, versatile, and easy-to-learn programming language.
- It was created by Guido van Rossum and released in 1991.
- Key uses include web development, data science, AI, automation, and games.
- Install Python by downloading from python.org and *crucially* selecting "Add Python to PATH."
- You can write and run Python code online using IDEs like Jupyter Notebook.
- Python code is built from fundamental units called Tokens: Keywords (reserved words like if, for), Identifiers (names you choose), Literals (fixed values like numbers, text), Operators (+, -, ==), and Punctuators (symbols).
- Variables are named containers for data; Python automatically handles their type (dynamic typing).
- Common Data Types are integers (int), decimal numbers (float), text (str), lists ([]), tuples (()), and dictionaries ({}).
- input() reads user entry as text; use int() or float() to convert it to numbers.
- Operators perform actions on data, including arithmetic (+, -, *, /), comparisons (==, >), and logical operations (and, or, not).
- Control structures dictate program flow:
- Conditional statements (if, elif, else) make decisions based on conditions.
- Loop statements (for) repeat code blocks, often for each item in a sequence.

Objective Type Questions

1. Name an online IDE to run Python programme
2. Which kind of token is 'finally'?
3. "Hello" is which kind of data type?
4. Which is called a named memory location?
5. Write output of the program of the code given below

   msg="Good"

   print(msg*4)
6. What is the first name of Python's developer?
7. In what year was Python first released?
8. What type of programming language is Python considered for rapid application development?
9. What is the term for fixed values like numbers or text in Python code?
10. What symbol is used for assignment in Python?
11. What type of operator is and?
12. What kind of loop iterates over a sequence?
13. Which keyword is used to define a function?
14. What is the data type for whole numbers?
15. What type of variables are accessible throughout the entire program?

Answers to Objective Type Questions

1. Jupyter
2. Keyword
3. String
4. Variable
5. GoodGoodGoodGood
6. Guido
7. 1991
8. Scripting

9. Literals
10. =
11. Logical
12. For
13. Def
14. Int
15. Global

Assignments

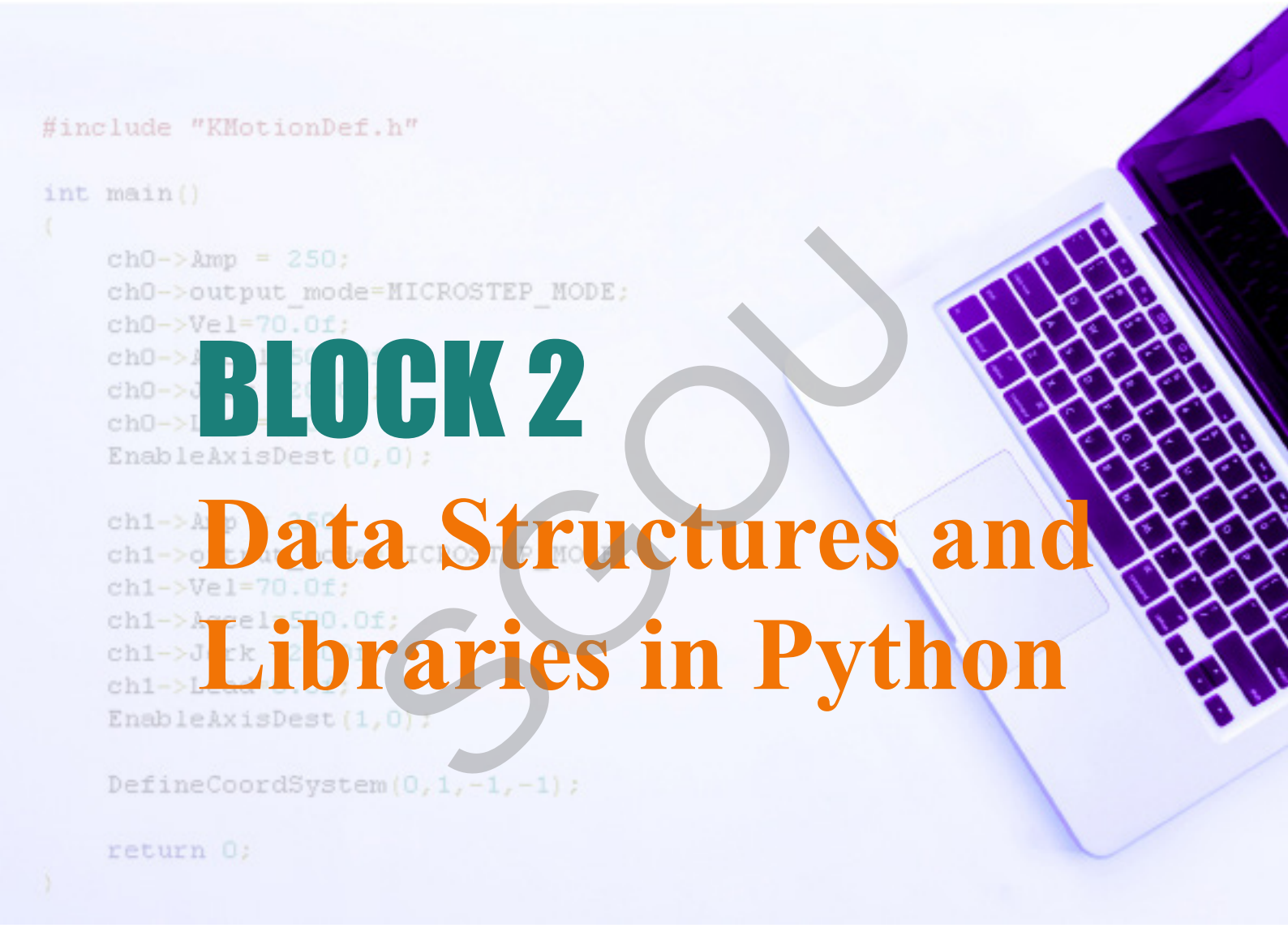
1. Explain applications of Python.
2. What are the character set used in Python
3. Write a program to print a message using online IDE
4. Write short note on different tokens used in Python
5. Explain different operators used in Python
6. What are the statements used in pythons
7. Write a program to find largest among three numbers

References

1. A Beginner's Guide To Learn Python In 7 Day, Author: Ramsey Hamilton
2. Python Programming for Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies). Author: James Tudor
3. https://www.python.org/about/gettingstarted/

Suggested Reading

1. Python online documents. https://docs.python.org/3/library/operator.html



BLOCK 2

Data Structures and Libraries in Python

Introduction to Data Structures

Learning Outcomes

At the end of this unit, the learner will be able to:

- define primitive and non-primitive data structures.
- list different types of built-in data structures in Python.
- identify the syntax used to create arrays, lists, tuples, dictionaries, and sets.
- recall commonly used methods for manipulating lists, strings, and dictionaries.
- name various string operations and their functions in Python.

Prerequisites

In your earlier programming experience, you've likely worked with values like names, numbers, or simple messages. These values are often stored using variables, which hold one item at a time. But when you want to manage a group of related data like the marks of several students or a list of city names, handling each item individually becomes difficult and time-consuming.

This is where Python offers useful data structures to make handling such data easier.

You may also know how to use basic data types such as integers, strings, and booleans. Building on this foundation, this unit will introduce data structures, special ways of storing and organizing data so that it can be used efficiently. You will explore both primitive types (like numbers and characters) and non-primitive types such as arrays, lists, tuples, dictionaries, sets, and strings.

These structures are useful for storing multiple values, accessing them quickly, and performing operations such as sorting, searching, adding, or removing elements. By the end of this unit, you will see how these tools can make your programs shorter, faster, and easier to understand. Understanding this unit will give you the power to handle real-world data better and is essential for further learning in Python programming.

Key Concepts

Arrays, Lists, Tuples, Dictionaries, Sets

Discussion

Data structures are fundamental building blocks for organizing and managing data in Python. They play a crucial role in determining the efficiency and performance of your programs.

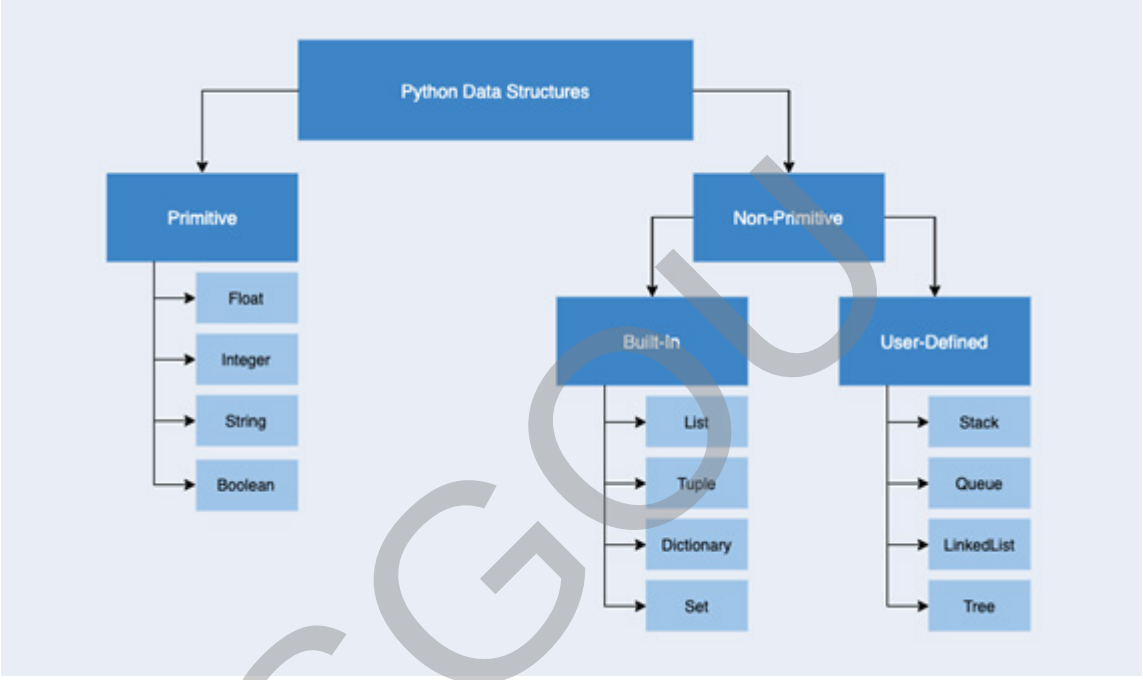


Fig 2.1.1 types of data structures

Figure 2.1.1 shows the types of data structures. Different data structures used in Python are:

- Primitive data structures
- Non-primitives

2.1.1 Primitive Data Structures

Primitive data structures are basic and fundamental data types provided by programming languages. They are directly operated upon by the machine and are built into the language itself. Examples of primitive data structures include integers, floating-point numbers, characters, booleans, and pointers.

2.1.2 Non-Primitive Data Structures

Non-primitive data types in Python are complex data structures that can store multiple values and support various operations for organizing and managing data. They are derived from primitive types and are used to handle more complex programming tasks. Non primitive data structures are two types:

- Built-in data structures
- User defined data structures

2.1.2.1 Built-in data types

Built-in data structures in programming languages are those that are provided as part of the standard library or core language features. In Python, several built-in data structures are readily available for developers to use without requiring additional libraries. Here are some of the most commonly used built-in data structures in Python are Arrays, Lists, Tuples, Dictionaries, Sets, Strings, Bytes and Byte Arrays.

2.1.2.2 User defined data types

User defined data structures in Python are data structures that are created by users based on their specific requirements. User defined data structures are implemented by the users themselves using classes and objects. These structures can encapsulate any type of data and define custom behavior and operations. Here are some examples of user defined data structures in Python, linked list, Stack, Queue etc.

2.1.3 Arrays

An array is a data structure that stores a collection of items, usually of the same type, such as all numbers or all strings. Instead of creating separate variables for each item (like mark1, mark2, mark3), we can store them in an array and access them using their index number.

For example, if we want to store 5 marks, we can use one array instead of 5 variables. This makes our program shorter, cleaner, and easier to work with, especially when handling large amounts of data.

Types of Arrays in Python

Python does not have traditional array data types like some other programming languages. However, it allows you to work with arrays in the following ways:

1. Using Lists (most common and flexible)
2. Using the array module (for numeric arrays only)
3. Using the NumPy library (for large, scientific or multi dimensional arrays)

2.1.3.1 Creation of Array

In Python, arrays can be created using the array module, which is useful when you need to store elements of the same data type efficiently (like integers or floats).

```
import array

numbers = array.array('i', [10, 20, 30, 40])

print(numbers)
```

output

```
array('i', [10, 20, 30, 40])
```

2.1.3.2 Indexing of Array

Indexing means accessing elements of an array using their position. In Python, array indexing starts from 0 (the first element is at index 0).

Example

```
import array
# Creating an array of integers
marks = array.array('i', [50, 60, 70, 80, 90])
# Accessing elements using indexing

print(marks[0])   # First element

print(marks[3])   # Fourth element

print(marks[-1])  # Last element (using negative index-negative indexing starts from the end)
```

Output

```
50

80

90
```

2.1.3.3 Array Operations

Here are the most commonly used operations and methods in arrays (especially using the array module):

1. Accessing Elements

You can get any item using its index:

```
print(numbers[0])  # First item
```

2. Adding Elements

- append(x) – Adds x at the end

  numbers.append(50)

  insert(i, x) – Inserts x at index i

  numbers.insert(2, 25)

3. Removing Elements

- remove(x) – Removes the first occurrence of x

  numbers.remove(20)

- pop() – Removes and returns the last item

  numbers.pop()

4. Modifying Elements

You can change an existing item like this:

```
numbers[1] = 99
```

5. Length of Array

To find out how many items are in the array:

```
len(numbers)
```

6. Finding Index

To know the position of a specific value:

```
numbers.index(30)
```

7. Reversing Array

To reverse the array in place:

```
numbers.reverse()
```

8. Looping Through an Array

To print or use each element in an array one by one, use a for loop:

```
for item in numbers:
print(item)
```

2.1.4 Lists

In Python, a variable usually stores only one piece of data. For example, if you write student_name = "KKG", the variable student_name holds just one value. But what if you need to store and work with many pieces of data, such as a list of scores or names? Creating separate variables like x1, x2, x3, and so on would be time consuming and hard to manage. To solve this problem, Python provides a data type called a list.

2.1.4.1 Creation of Lists

A list allows you to store multiple values under a single variable name. It is written using square brackets [], with items separated by commas. For example, x = [20, 100, 30, 19] stores four values in one variable. This makes your code easier to read and manage, especially when working with large amounts of data.

Lists are very flexible. They can store items of different types, such as numbers, strings, or even another list. However, it's common to keep items of the same type. Lists are also mutable, which means you can change the values, add new items, or remove items after the list has been created.

Items in a list are accessed using indexes. In Python, the index starts at 0, not 1. For example, if you have a list called Age = [2, 4, 1, 10, 5], then Age[0] gives you 2, and Age[1] gives you 4. The last item in the list is Age[4] = 5, since the list has five items and the last index is always one less than the number of items.

You can also use expressions inside the index brackets. For instance, if i = 3, then Age[i + 1] gives you the value at Age[4]. Python will evaluate the expression inside the brackets before using it to find the item in the list.

A list can contain a mix of data types. For example, L = ['Covid', 2.0, 5, [10, 20]] is a list that holds a string, a float, an integer, and another list. If a list has no items at all, it is called an empty list, and it is written as [].

In the following programs guess the output first, then run the code and check the result.

Program 1: Write and Run the below code and see the result

```
lis= []
print(lis)
```

Output : []

Program 2: Run the below code and observe the result.

```
Student_List= ["John", "KKG", "Jane"]
print(Student_List)
```

Output: ['John', 'KKG', 'Jane']

Program 3: Write and Run the below code and check the result

```
Student_List= ["John", "KKG", "Jane"]
 print(Student_List[0])
```

Output: John

2.1.4.2 List Indexing

Python lists are ordered collections, which means each element has a position or index starting from 0. Indexing and slicing are techniques used to access parts of a list.

Syntax: list[index]

- Positive indexing starts from 0
- Negative indexing starts from -1 (last item)

Indexing returns the item as shown below.

Example:

```
Student_List=[ “John”, “KKG”, “Jane”]
Student_List[0]
```

Output : John

Student_List[1]

Output : KKG

Student_List[-1]

Output : Jane

Student_List[-2]

Output : KKG

2.1.4.3 Methods used for List manipulation

List manipulation means changing or working with the list using different actions like adding, removing, updating, or accessing elements. List methods are used to perform various operations on a list.

- **append() :**

You can add new items at the end of the list, by using the append() method. For example,

Student_List= ["John", "KKG", "Jane"]

Student_List.append("Shan")

print(Student_List)

Output : ['John', 'KKG', 'Jane', 'Shan']

- **insert(i, item)**

Insert an item at a given position(index). The first argument is the index of the element before which to insert, so StudentList.insert(1, x) inserts at the front of the list. For example,

Student_List= ["John", "KKG", "Jane"]

Student_List.insert(1,"Sam")

print(Student_List)

Output : ['John', 'Sam', 'KKG', 'Jane']

- **list.remove(x)**

Remove the first item from the list whose value is equal to x.

Student_List= ["John", "KKG", "Jane"]

Student_List.remove("KKG")

print(Student_List)

Output : ['John', 'Jane']

- **list.pop([i])**

Remove the item at the given position in the list, and return it. If no index is specified, pop() removes and returns the last item in the list.

Student_List = [“John”,”KKG”,”Jane”]

print(Student_List.pop(1))

Output : KKG

- **list.clear()**

Remove all items from the list.

- **list.count(x)**

Return the number of times x appears in the list.

Student_List= ["John", "KKG", "Jane", "John"]

print(Student_List.count("John"))

Output : 2

- **list.reverse()**

Reverse the elements of the list in place

Student_List= ["John", "KKG", "Jane"]

Student_List.reverse() print(Student_List)

Output : ['Jane', 'KKG', 'John']

- **list.sort()**

Sort the items of the list in place

Student_List= ["John", "KKG", "Jane"]

Student_List.sort()

print(Student_List)

Output : ['Jane', 'John', 'KKG']

2.1.5 Tuples

A tuple in Python is an ordered, immutable collection of items that allows duplicates. This means the elements in a tuple maintain their position and cannot be changed once defined. Tuples can store elements of different data types such as integers, strings, or even other tuples. Duplicate values are allowed in a tuple.

2.1.5.1 Creation of Tuple

Tuples are created by placing items inside parentheses (), separated by commas.

Example:

```
my_tuple = (10, 20, 30)

colors = ("red", "green", "blue")

info = ("Alice", 25, "Engineer")
```

Key Characteristics of Tuples:

- **Ordered:** The position of each element is fixed.
- **Immutable:** Once created, you cannot add, remove, or change elements.
- **Allow duplicates:** You can have the same value appear more than once, e.g. (1, 1, 2).

Example 1: Printing the Entire Tuple

```
cars=("Toyota", "KIA", "Maruthi")

print(cars)
```

Output : Toyota, KIA, Maruthi

Example 2: Accessing an Element by Index

```
cars=( "Toyota", "KIA",  "Maruthi")

print(cars[1])
```

Output : KIA (remember the index starting from zero)

Items in tuples are unchangeable, which means we cannot change or add. To add an item to a tuple, convert the tuple into the python list, add a new item and convert it back to a tuple.

2.1.5.2 Tuple Indexing

Indexing is the way we access individual elements in a tuple using their position. In Python, indexing starts at 0, meaning the first element is at position 0, the second at position 1, and so on.

Example:

```
fruits = ("apple", "banana", "cherry", "date")

print(fruits[0])

print(fruits[2])
```

Output:

apple

cherry

In the example above:

- fruits[0] accesses the first element, which is "apple".
- fruits[2] accesses the third element, which is "cherry".

2.1.5.3 Manipulation of Tuples

Tuples are immutable, which means you cannot change, add, or remove items once the tuple is created. However, you can still access, slice, iterate, and even perform certain operations like joining or searching within them.

Common Tuple Manipulations

1. Accessing Elements

Tuple items are accessed using index numbers starting from 0. Negative indexing can be used to access elements from the end of the tuple.

```
colors = ("red", "green", "blue")
print(colors[0])
```

Output: red

```
print(colors[-1])
```

Output: blue

2. Slicing Tuples

You can slice a tuple to get a portion of it:

```
numbers = (10, 20, 30, 40, 50)
print(numbers[1:4])
```

Output: (20, 30, 40)

3. Looping Through a Tuple

```
fruits = ("apple", "banana", "cherry")
for fruit in fruits:
print(fruit)
```

Output:

```
apple
banana
cherry
```

4. Checking for an Item

Use in to check if an item exists:

```
print("banana" in fruits)
```

Output: True

5. Joining Tuples

You can concatenate two tuples:

t1 = (1, 2, 3)

t2 = (4, 5)

t3 = t1 + t2

print (t3)

Output: (1, 2, 3, 4, 5)

6. Counting and Indexing

a. Count():

To find the number of times a specified word is repeated.

car= ("Maruthi", "Toyota", "Maruthi", "Mahindra", "Maruthi")

x=car.count("Maruthi")

print(x)

Output : 3

b.Index():

To find the position or index of an item in a tuple.

In the following example, the tuple has 4 items. The value Mahindra is repeated two times. The index method will display the first occurrence. The index of Maruti is zero. The index of the first occurrence of Mahindra is 1.

car = ("Maruti", "Mahindra", "Toyota", "Mahindra")

print("Index of Maruti:", car.index("Maruti")) # Output: 0

print("Index of Mahindra:", car.index("Mahindra")) # Output: 1 (first occurrence)

2.1.6 Dictionary

Another useful data type in Python is the dictionary. A dictionary stores data in the form of key value pairs, where each key is unique and used to access its corresponding value. The keys must be of an immutable type, such as strings or numbers, while the values can be of any data type. Dictionaries are created using curly braces {}, and an empty dictionary is simply written as {}. For example, a student's name and age can be stored as {"name": "John", "age": 16}. Dictionaries are ordered, meaning the items keep the order in which they were added. Unlike lists, where data is accessed by position (index), in dictionaries, data is accessed using the key. The main operations include storing a value with a key and retrieving a value by providing its key.

Example 1: Display a Dictionary

Student_Phone= {"KKG": 8608754, 'John': 890744}

print(Student_Phone)

Output: {'KKG': 8608754, 'John': 890744}

Example 2: Access a Value Using a Key

In the following example, the dictionary names cars and stores the car manufacturer name and country. The manufacturer's name is the key and country names are the value.

```
cars ={ "Maruthi" : "India", "Toyota":"Japan", "KIA":"Korea"}

x = cars.get("KIA")

print(x)
```

Output : Korea

Example 3: Display all the keys from a dictionary

```
cars ={"Maruthi":"India","Toyota":"Japan","KIA":"China"}

print(cars.keys())
```

Output : dict_keys(['Maruthi', 'Toyota', 'KIA'])

Example 4: Add a new item to a dictionary

```
Student_phone = {"KKG": 8608754, 'John': 890744}

Student_phone ["Shan"] = 989643

print(student_phone)
```

The first step is to store two people's phone numbers. (Note: In this example only two phone numbers, it could be any number of names and phone numbers). The second statement will add another person's phone number and the third statement will print all values from the dictionary as shown below:

{'KKG': 8608754, 'John': 890744, 'Shan': 989643}

2.1.6.1 Dictionary built-in methods

Python provides several built-in methods to work with dictionaries. The following are some of the Dictionary methods:

1. update():

Used to add a new key-value pair or update an existing one.

```
car= {"Maruthi":"2004", "Toyota": "2008 ", "Mahindra": "2007 "}
print(car)
car.update({"KIA":"2010"})
print(car)
```

Output :

{'Maruthi': '2004', 'Toyota': '2008 ', 'Mahindra': '2007 '}

{'Maruthi': '2004', 'Toyota': '2008 ', 'Mahindra': '2007 ', 'KIA': '2010'}

2.clear():

Remove all items from the dictionary and return an empty dictionary.

```
car= {"Maruthi": "2004", "Toyota": "2008 ", "Mahindra": "2007 "}
car.clear()
print(car)
```

Output : {}

3.copy():

Creates and returns a **copy** of the dictionary.

```
car= {"Maruthi": "2004", "Toyota": "2008 ", "Mahindra": "2007 "}
x= car.copy()
print(x)
```

Output : {'Maruthi': '2004', 'Toyota': '2008 ', 'Mahindra': '2007 '}

4.keys() :

To return all the keys used in a dictionary.

```
car= {"Maruthi": "2004", "Toyota": "2008 ", "Mahindra": "2007 "}
x= car.keys()
print(x)
```

Output : dict_keys(['Maruthi', 'Toyota', 'Mahindra'])

5.Values():

To return the values of a dictionary

```
car= {"Maruthi": "2004", "Toyota": "2008 ", "Mahindra": "2007 "}
x= car.values()
print(x)
```

Output : dict_values(['2004', '2008 ', '2007 '])

6. item():

To access both keys and values from a dictionary

```
car = {"Maruthi": "2004", "Toyota": "2008", "Mahindra": "2007"}
for brand, year in car.items():
print(brand, year)
```

Output :

Maruthi 2004

Toyota 2008

Mahindra 2007

2.1.7 Sets

Sets are an unordered collection of unique elements in Python. They are defined using curly braces {} and offer several advantages over other data structures.

Example:

```
my_set = {1, 2, "apple", True}
print(my_set)
```

Output:{1, 2, 'apple'}

(Python considers True as 1, so adding both 1 and True results in only one value being stored. Sets do not allow duplicates, and they are unordered, meaning the items may appear in any order when printed.)

2.1.7.1 Set Built-in Methods

The following are some of the set methods available in Python.

1. clear() :

To delete all elements from list

```
car= ["Maruthi", "Toyota", "Mahindra"]
print(car)
```

Output : ['Maruthi', 'Toyota', 'Mahindra']

```
car= ["Maruthi", "Toyota", "Mahindra"]
car.clear()
print(car)
```

Output : []

2. add() :

To add an item to set

```
car= {"Maruthi", "Toyota", "Mahindra"}
car.add("Kia")
print(car)
```

Output : {'Toyota', 'Kia', 'Mahindra', 'Maruthi'}

3.copy():

To make a copy of a set.

```
car= {"Maruthi", "Toyota", "Mahindra"}
Newcar = car.copy()
print(Newcar)
```

Output : {'Toyota', 'Mahindra', 'Maruthi'}

4.difference():

To return a set that contains the difference between two sets.

```
m = n.difference(x)
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.difference(y)
print(z)
```

Output : {'cherry', 'banana'}

5.union() :

Return the union of two sets.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.union(y)
print(z)
```

Output : {'google', 'microsoft', 'apple', 'cherry', 'banana'}

6.discard():

To remove an item from a set.

```
fruits={"apple", "banana", "cherry"}
fruits.discard("banana")
print(fruits)
```

Output : {'apple', 'cherry'}

2.1.8 String manipulations in python

String manipulation refers to modifying or processing text (strings) to achieve certain results. It includes tasks like changing cases, finding words, replacing text, combining or splitting strings, and formatting messages. Python makes this easy with many built-in string methods.

Immutability of Strings

In Python, strings are immutable. This means:

- You cannot change a string after it is created.
- Any operation on a string returns a new string.

Example:

If text = "hello" and you try text.upper(), it gives "HELLO" but does not change the original text.

2.1.8.1 Basic Operations

Strings in Python are sequences of characters. Several basic operations can be performed on strings such as creation, accessing, joining, slicing, and repeating.

1. Creating Strings

Strings can be enclosed in:

- Single quotes: 'Hello'
- Double quotes: "Hello"

Example:

```
greeting = 'Hello, world!'
```

2. Accessing Characters

Each character in a string has a position (index) starting from 0.

Example:

```
text = "Python"
text[0]        Output: 'P'
text[1]        Output: 'y'
```

You can also extract part of a string using slicing:

```
text[0:3]
```

Output: 'Pyt' (from index 0 to 2)

3. Concatenation (Joining Strings)

You can join strings using the + operator or the join() method.

Example:

```
first="Sree"
last="narayana"
full_name = first + " " + last
```

Output: Sree narayana

or

```
full_name = " ".join([first, last])
```

Output: Sree narayana

4. Repetition

You can repeat a string using the * operator.

Example:

```
text = "Python"
text * 3
```

Output: Python Python Python

2.1.8.2 String Methods

Python provides a variety of built-in string methods to perform operations such as case conversion, searching, splitting, joining, replacing, trimming, and formatting.

1. Case Conversion

Table 2.1.1 Case conversion methods

Method	Description
text.upper()	Converts all characters to uppercase
text.lower()	Converts all characters to lowercase
text.capitalize()	Capitalizes the first letter only
text.title()	Capitalizes the first letter of every word

2. Searching in Strings

Table 2.1.2 Searching methods

Method	Description
text.find("word")	Returns index of first match, or -1 if not found
text.index("word")	Like find(), but gives an error if not found
text.startswith("Hello")	Returns True if the string starts with "Hello"
text.endswith("world")	Returns True if the string ends with "world"

3. Splitting and Joining

- split() – Breaks a string into a list using spaces (or other characters)

  Example:

  "Python is fun".split() gives ['Python', 'is', 'fun']

- join() – Combines elements of a list into one string

  Example:

  "-".join(['Python', 'is', 'fun']) gives 'Python-is-fun'

4. Replacing Text

- text.replace("old", "new") – Replaces old text with new text

  Example:

  "I like Java".replace("Java", "Python") → 'I like Python'

5. Trimming Whitespace

Table 2.1.3 Methods for trimming white spaces

Method	Description
strip()	Removes spaces from both ends
lstrip()	Removes spaces from the left
rstrip()	Removes spaces from the right

6. Formatting Strings

Using format():

"Hello, {}!".format("Alice")

Output: 'Hello, Alice!'

Using f-strings (Python 3.6+):

name = "Alice"

f"Hello, {name}!"

Output: 'Hello, Alice!'

Recap

- Data structures are used to store and organize data efficiently in Python.
- Two main types of data structures:
- Primitive data types: integers, floats, characters, etc.
- Non-primitive data types: built-in and user-defined structures.
- Arrays are used to store items of the same data type.
- Created using the array module.
- Elements are accessed using index numbers.
- Common operations include adding (append()), inserting (insert()), removing (pop()), and updating elements.
- Lists can store items of different data types.
- Created using square brackets [].
- Lists are ordered and mutable (can be changed).
- Support operations like adding, removing, sorting, and reversing elements.
- Tuples are similar to lists but are immutable (cannot be changed).
- Created using parentheses ().
- Elements are accessed using indexing.
- Allow duplicate values and support slicing and joining.
- Dictionaries store data in key-value pairs.
- Created using curly braces {}.
- Values are accessed using their keys.
- Support operations like adding, updating, copying, and retrieving all keys or values.
- Sets store unordered, unique items.
- Created using curly braces {}.
- Do not allow duplicate values.
- Common operations include adding items (add()), removing (discard()), and finding union or difference between sets.

Objective Type Questions

1. Which data structure stores data as key-value pairs?
2. What is the method used to add an element at the end of a list?
3. Which data structure in Python is ordered and immutable?
4. Which data type is used to store unique and unordered items?
5. What is the indexing position of the first element in a Python list?
6. Which built-in module is used to create arrays in Python?
7. What function is used to find the number of items in a list or array
8. What is the term for accessing the last element using negative indexing?
9. Which operator is used to join two strings?
10. Which function returns all the keys in a dictionary?
11. Which data structure can be created using square brackets?

Answers to Objective Type Questions

1. Dictionary
2. append
3. Tuple
4. Set
5. Zero
6. array
7. len
8. -1
9. +
10. keys
11. List

Assignments

1. Explain the difference between primitive and non-primitive data structures in Python. Give suitable examples.

2. Write a Python program to create an array using the array module. Perform the following operations: append, insert, and remove elements.

3. Create a list in Python with 5 fruits. Show how to add a new fruit, remove one fruit, and print all the items using a loop.

4. Create a dictionary in Python with names of 3 people and their phone numbers. Then add one more entry and print all names and phone numbers.

5. Write a Python program to take a sentence as input and perform the following operations:

   - Convert to uppercase
   - Replace one word with another
   - Split the sentence into words

References

1. A Beginner's Guide To Learn Python In 7 Day, Author: Ramsey Hamilton

2. Python Programming for Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies). Author James Tudor

3. https://www.python.org/about/gettingstarted/

Suggested Reading

1. Python online documents.https://docs.python.org/3/library/operator.html

Libraries

Learning Outcomes

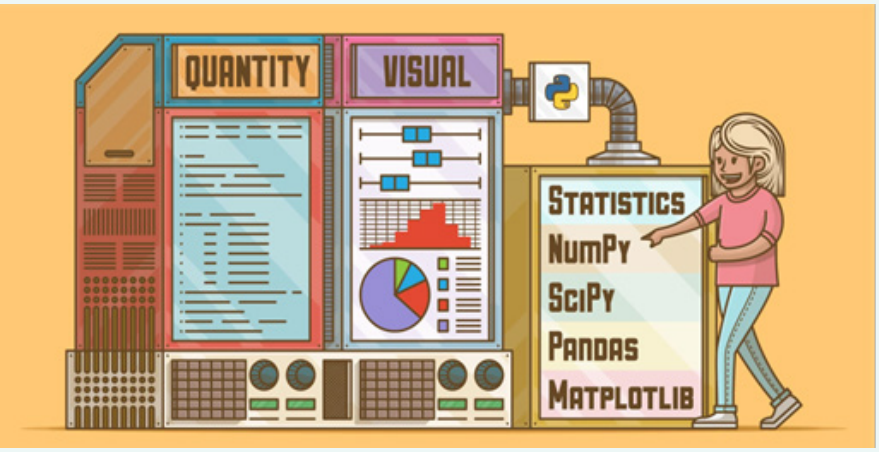
The students will be able:

- list commonly used Python libraries for scientific computing and data analysis (e.g., NumPy, Pandas, Matplotlib, SciPy, Scikit-learn).
- define what an N-dimensional array is in NumPy.
- recall the different data types supported by NumPy arrays.
- identify basic array attributes such as shape, size, and dimensions and various array operations.
- name the functions used for creating arrays from existing data and generating numerical ranges in NumPy.
- recognise the basic features and purposes of Pandas, Matplotlib, SciPy, and Scikit-learn.

Prerequisites

To grasp the full scope of Python's data-centric libraries (NumPy, Pandas, Matplotlib, SciPy, and Scikit-learn), learners must first understand basic Python programming concepts. This includes data types, variables, lists and functions. These fundamentals are essential because most libraries are built on top of Python's core syntax and structures. For example, before working with a NumPy array, a student should be comfortable creating and manipulating Python lists. Imagine tracking a week's worth of daily expenses, one must know how to store those values in a list and calculate the total. This basic skill translates directly to using arrays and applying mathematical operations in NumPy.

Next, a solid foundation in mathematics and data handling is crucial for effective use of libraries like NumPy and SciPy. Learners should understand arrays, indexing, slicing, and numerical operations, as these are used heavily in scientific computing and data processing. For example, in a weather monitoring system, slicing out temperature readings for a specific week from a month-long dataset and computing the average is a common task.



Lastly, familiarity with data visualization concepts, especially for Pandas, Matplotlib, and Scikit-learn. For example, analyzing sales data using Pandas involves reading the file, filtering specific columns, and generating summary statistics. Matplotlib then helps visualize trends through graphs like bar charts or line plots. When using Scikit-learn, learners should be aware of structured datasets, data preprocessing steps, and basic supervised learning ideas.

Keywords

NumPy, Ndarray, Pandas, Matplotlib, SciPy, Scikit-learn

Discussion

2.2.1 Basic concepts of Libraries

Python is a popular programming language with a fast expanding global user base. It is used to create applications for a variety of domains, including software engineering, data science, machine learning, and more. It may also handle common problems. Python is becoming more and more popular because of its extensive libraries, which are collections of codes that we may utilize in our programs to perform specific tasks. Additionally, Python libraries provide classes, documentation, values, message templates, and configuration information.

A library often consists of a collection of books or a space where numerous volumes are kept for future use. Similarly, a library in the context of programming is a group of precompiled programs that can be utilized for certain well-defined operations later on in a program. A library may contain classes, values, message templates, configuration information, documentation, and other materials in addition to pre-compiled codes.

A **Python library** is a grouping of related modules that are connected to one another. Because developers don't have to write the same code for multiple projects, it has a code bundle that they can use frequently in numerous programs, which speeds up and

streamlines programming. In addition, every Python library has its own source code. Various fields, such as computer science, data visualization, and machine learning, employ Python libraries. Python's precise tokens, syntax, and semantics are included in the Python Standard Library. Basic system functions like I/O and a few other fundamental modules are accessible through built-in modules. The majority of Python libraries are developed in C. Over 200 essential modules make up the Python standard library. The combination of these factors makes Python a high-level programming language. The role of the Python Standard Library is crucial. Python's features are inaccessible to programmers without it. A programmer's life is made easier by a number of additional Python libraries, though. Let's examine some of the **libraries** that are frequently used:

1. **NumPy:** The name "NumPy" stands for "Numerical Python". It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use NumPy internally to perform several operations on tensors. Array Interface is one of the key features of this library.

2. **Pandas:** Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.

3. **Matplotlib:** This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.

4. **SciPy:** The name "SciPy" stands for **"Scientific Python"**. It is an open-source library used for high-level scientific computations. This library is built over an extension of NumPy. It works with NumPy to handle complex computations. While NumPy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.

5. **Scikit-learn:** It is a famous Python library for working with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with NumPy and SciPy.

6. **TensorFlow:** This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.

7. **Scrapy:** It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data.

8. **PyGame:** This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with the Python programming language.

9. **PyTorch:** PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.

10. **PyBrain:** The name "PyBrain" stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that's why it is really helpful for developers that are new in research fields.

2.2.1.1 How does a Library work?

By importing a library into a Python script, users can easily access its functions and classes, which are stored in a specific module. When a function from a library is called, Python loads the necessary code into memory and executes it, providing the result. This modular approach makes Python programming more efficient and flexible, enabling developers to build powerful applications with minimal effort. Basic steps needed for the working of a Python library are:

Import the Library: Use the **import** keyword to load the library into your Python script.

Example: import math

Access Library Features: Use **dot notation** to call functions or use classes from the library.

```
result= math.sqrt(25)

print(result)
```

Output

```
5.0
```

Execution: When a function is called, Python internally runs the code written in the library and returns the result.

2.2.2 NumPy

NumPy (Numerical Python) is a fundamental library for Python numerical computing.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. It provides efficient multi-dimensional array objects and various mathematical functions for handling large datasets making it a critical tool for professionals in fields that require heavy computation. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.

2.2.2.1 Key Features

NumPy provides a range of powerful features that make numerical and scientific computing in Python fast, efficient, and easy to use. It's various characteristics that make it popular over lists. Fundamental features of NumPy are:

- **N-Dimensional Arrays:** NumPy's core feature is ***ndarray***, a powerful N-dimensional array object that supports homogeneous data types.
- **Arrays with high performance:** Arrays are stored in contiguous memory locations, enabling faster computations than Python lists.
- **Broadcasting:** This allows element-wise computations between arrays of different shapes. It simplifies operations on arrays of various shapes by automatically aligning their dimensions without creating new data.
- **Vectorization:** Eliminates the need for explicit Python loops by applying operations directly on entire arrays.
- **Linear algebra:** NumPy contains routines for linear algebra operations, such as matrix multiplication, decompositions, and determinants.

2.2.3 DataTypes in NumPy

A data type in **NumPy** is used to specify the type of data stored in a variable. The below table depicts the list of characters available in NumPy to represent data types (Table: 2.2.1).

Table 2.2.1: List of characters available in NumPy

Character	Meaning
b	Boolean
f	Float
m	Time Delta
O	Object
U	Unicode String
i	Integer
u	Unsigned Integer
c	Complex Float
M	DateTime
S	String
V	A fixed chunk of memory for other types (void)

NumPy supports a wide range of data types that define the kind of elements stored in an array, such as integers, floats, complex numbers, and more. The list of various types of data types (Table 2.2.2) provided by NumPy are given below:

Table 2.2.2: Data types provided by NumPy

Data type	Description
bool_	Boolean
int_	Default integer type (int64 or int32)
intc	Identical to the integer in C (int32 or int64)
intp	Integer value used for indexing
int8	8-bit integer value (-128 to 127)
int16	16-bit integer value (-32768 to 32767)
int32	32-bit integer value (-2147483648 to 2147483647)
int64	64-bit integer value (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned 8-bit integer value (0 to 255)
uint16	Unsigned 16-bit integer value (0 to 65535)
uint32	Unsigned 32-bit integer value (0 to 4294967295)
uint64	Unsigned 64-bit integer value (0 to 18446744073709551615)
float_	Float values
float16	Half precision float values
float32	Single-precision float values
float64	Double-precision float values
complex_	Complex values
complex64	Represent two 32-bit float complex values (real and imaginary)
complex128	Represent two 64-bit float complex values (real and imaginary)

2.2.3.1 Checking the Data Type of an Array

The datatype of the NumPy array can be identified by using ***dtype.*** Then it returns the data type of all the elements in the array.

Example

```
import numpy as np

# Create a NumPy array

arr = np.array([1, 2, 3, 4, 5])

# Check the data type of the array

data_type = arr.dtype

print(data_type)
```

Output

```
int64
```

2.2.3.2 Create arrays with a Defined Data Type

To create an array with a defined data type by specifying the **"dtype"** attribute in **numpy.array()** method while initializing an array.

Example

```
import numpy as np

arr1 = np.array([1, 2, 3, 4], dtype=np.float64)

# Print the arrays and their data types

print(arr1.dtype)
```

Output

```
float64
```

2.2.4 Arrays in NumPy

Arrays in NumPy form the backbone of numerical computing in Python. Unlike regular Python lists, NumPy arrays (known as ndarray) offer powerful features such as multidimensional structure, fixed size, and support for a wide range of mathematical operations that are both fast and memory-efficient. NumPy arrays enable vectorized computations, which means operations can be applied on entire datasets without the need for explicit loops. This makes NumPy an essential tool for data analysis, machine learning, scientific computing, and engineering applications.

2.2.4.1 Array attributes

NumPy arrays (ndarray objects) come with several useful **attributes** that help you understand and manipulate their structure and content. The most commonly used array attributes are listed below:

1. **Number of Dimensions (.ndim):** Returns the number of axes (dimensions) of the array.

   Example

   a = np.array([[1, 2, 3], [4, 5, 6]])

   print(a.ndim)

   Output

   2

2. **Dimensions of the Array (.shape):** Returns a tuple indicating the size of the array in each dimension.

   Example

   a = np.array([[1, 2, 3], [4, 5, 6]])

   print(a.shape)

   Output

   (2, 3) ***# 2 rows, 3 columns***

3. **Total Number of Elements (.size):** Gives the total number of elements in the array.

   Example

   Output

   a = np.array([[1, 2, 3], [4, 5, 6]])

   print(a.size)

   6

4. **Data Type of Elements (.dtype):** Shows the data type of the elements stored in the array.

   Example

   a = np.array([[1, 2, 3], [4, 5, 6]])

   print(a.dtype)

   Output

   int64 ***#may vary by system***

2.2.4.2 One Dimensional Array

A **one-dimensional array** (Fig 2.2.1) in NumPy is essentially a sequence or list of elements, all of which are of the same type. It is a type of linear array. It is similar to a Python list, but offers the benefit of faster operations and a more compact memory footprint. A 1D array as a vector, where each element is accessed by its index.

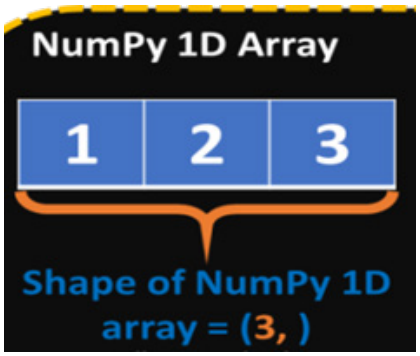


Fig 2.2.1 One Dimensional Array

Example

```
import numpy as np
# From a Python list
a = np.array([1, 2, 3, 4, 5])
print(a)
```

Output

```
[1 2 3 4 5]
```

2.2.4.3 Two Dimensional Array

A **two-dimensional array** (2D array) in NumPy is essentially a grid or matrix, where elements are arranged in rows and columns (Fig 2.2.2). This structure can represent things like images, tables of data, and more complex mathematical objects. Data is stored in tabular form.

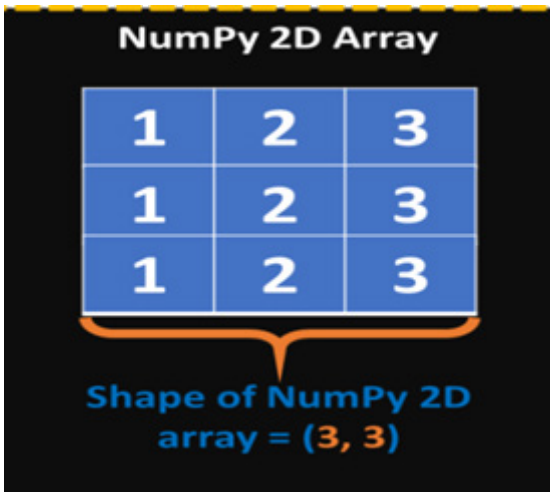


Fig 2.2.2 Two Dimensional Array

Example

```
import numpy as np
# Create a 2D array (3 rows, 4 columns)
a = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
print(a)
```

Output

```
[[1, 2, 3, 4],
[5, 6, 7, 8],
[9, 10, 11, 12]]
```

2.2.4.4 N- Dimensional Array

ndarray (short for *N-dimensional array*) is a core object in NumPy. It is a homogeneous array which means it can hold elements of the same data type. An N-dimensional array (or ndarray) in NumPy is an array with N dimensions, where N can be any non-negative integer. While 1D and 2D arrays are the most commonly used, NumPy allows you to work with arrays of higher dimensions (3D, 4D, etc.) to represent more complex data structures like tensors, volumetric data, or time-series data in multiple dimensions (Fig 2.2.3).

The key advantage of using N-dimensional arrays is that they allow you to work efficiently with large datasets in various fields such as image processing, machine learning, and scientific simulations.

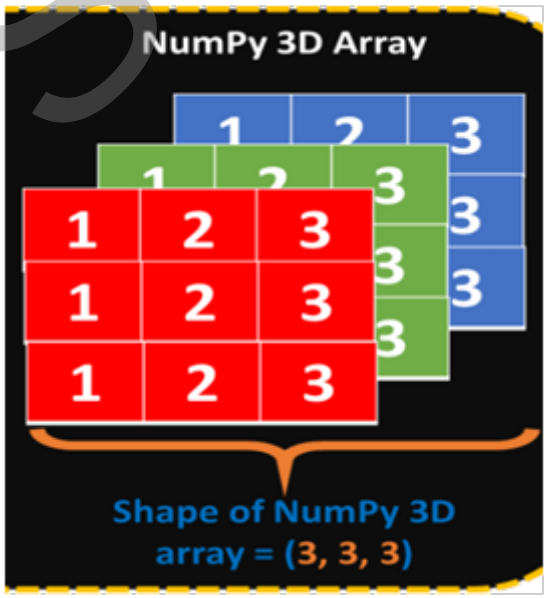


Fig 2.2.3 N-dimensional Array

Example

```
import numpy as np
#1D array
arr1 = np.array([1, 2, 3, 4, 5])
#2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
#3D array
arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(arr1)
print(arr2)
print(arr3)
```

Output

```
[1 2 3 4 5]
[[1 2 3]
[4 5 6]]
[[[1 2]
 [3 4]]
 [[5 6]
 [7 8]]]
```

2.2.4.5 Array from existing data

NumPy's ability to work fast and perform complex operations on arrays is really important in fields like data handling and performing scientific calculations. You can create arrays from existing data in NumPy by initializing NumPy arrays using data structures that already exist in Python, or can be converted to a format compatible with NumPy. Following are a few common ways to achieve this:

1. Using numpy.asarray() Function
2. Using numpy.frombuffer() Function
3. Using numpy.fromiter() Function
4. From Python Lists
5. From Python Tuples

1. Using numpy.asarray() Function

The **numpy.asarray()** function is used to convert various Python objects into NumPy

arrays. These objects include Python lists, tuples, other arrays, and even scalar values. This function ensures that the result is always a NumPy array, making it convenient for data manipulation and numerical computations.

Syntax:

numpy.asarray(a, dtype=None, order=None)

where,

- **a :** It is the input data, which can be a list, tuple, array, or any object that can be converted to an array.
- **dtype (optional) :** Desired data type of the array. If not specified, NumPy determines the data type based on the input.
- **order (optional) :** Specifies whether to store the array in row-major (C) or column-major (F) order. Default is None, which means NumPy decides based on the input.

Example

```
import numpy as np
# Convert list to array
list = [1, 2, 3, 4, 5]
arr_list = np.asarray(list)
print("Array from list:",arr_list)
```

Output

```
Array from list: [1 2 3 4 5]
```

2. Using numpy.frombuffer()Function

The numpy.frombuffer() function creates an array from a buffer object, such as bytes objects or byte arrays. This is useful when working with raw binary data or memory buffers. This function interprets the buffer object as one-dimensional array data. It allows you to specify the data type of the elements in the resulting array.

Syntax:

numpy.frombuffer(buffer, dtype=float, count=-1, offset=0)

where,

buffer − It is the buffer object containing the data to be interpreted as an array.

dtype (optional) − It is the desired data type of the elements in the resulting array. Default is float.

count (optional) − It is the number of items to read from the buffer. Default is -1, which means all data is read.

offset (optional) − It is the starting position within the buffer to begin reading data. Default is 0.

Example

```
import numpy as np
# Create bytes object
bytes = b'hello world'
# Create array from bytes object
arr_bytes = np.frombuffer(bytes, dtype='S1')
print("Array from bytes object:",arr_bytes)
```

Output

```
Array from bytes object: [b'h' b'e' b'l' b'l' b'o' b' ' b'w' b'o' b'r' b'l' b'd']
```

The dtype='S1' means that each element in the resulting NumPy array will be a **string of length 1** specifically, a **byte string** (not a Unicode string). This breakdown as :

- 'S' stands for a **byte string** (i.e., raw bytes, similar to b'a', b'b', etc.).
- '1' indicates the string length ;in this case, 1 byte per element.

3. Using numpy.fromiter() Function

The numpy.fromiter() function creates a new one-dimensional array from an iterable object. It iterates over the iterable object, converting each element into an array element.

Syntax:

```
numpy.fromiter(iterable, dtype, count=-1)
```

where,

- iterable − The iterable object that yields elements one by one.
- dtype − The data type of the elements in the resulting array.
- count (optional) − The number of items to read from the iterable. Default is -1, which means all items are read.

Example

```
import numpy as np
# Generator function that yields numbers
def generator(n):
  for i in range(n):
    yield i
# Create array from generator
```

```
gen_array=np.fromiter(generator(5), dtype=int)

print("Array from generator:",gen_array)
```

Output

```
Array from generator: [0 1 2 3 4]
```

4. From Python Lists

One of the most common ways to create a NumPy array is by converting a Python list. This method provides the numpy.array() function or numpy.asarray() function to convert lists, which are commonly used data structures in Python, into NumPy arrays.

Syntax:

```
numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
```

where,

- object − The input data, which in this case is a Python list.
- dtype (optional) − Desired data type of the array. If not specified, NumPy interprets the data type from the input data.
- copy (optional) − If True, ensures that a copy of the input data is made. If False, avoid unnecessary copies when possible.
- order (optional) − Specifies the memory layout order of the array. 'C' for row-major (C-style), 'F' for column-major (Fortran-style), and 'K' for the layout of the input array (default).
- subok (optional) − If True, subclasses are passed through; otherwise, the returned array will be forced to be a base-class array.
- ndmin (optional) − Specifies the minimum number of dimensions the resulting array should have.

Example

```
import numpy as np

# Convert list to array

list = [1, 2, 3, 4, 5]

arr_list = np.array(list)

print("Array from list:",arr_list)
```

Output

```
Array from list: [1 2 3 4 5]
```

5. From Python Tuples

Python tuples are another commonly used data structure that can be converted into

NumPy arrays. Like lists, tuples can be used to store multiple items, but they are immutable, meaning their content cannot be changed after creation. It can be used to represent both one-dimensional and multi-dimensional data using the **numpy.array()** function.

Example

```
import numpy as np
# Convert tuple to array
tuple = (1, 2, 3, 4, 5)
arr_tuple = np.array(tuple)
print("Array from tuple:",arr_tuple)
```

Output

```
Array from tuple: [1 2 3 4 5]
```

2.2.4.6 Array from numerical ranges

Creating arrays from numerical ranges in NumPy refers to generating arrays that contain sequences of numbers within a specified range. NumPy provides several functions to create such arrays, they are as follows:

1. Using numpy.arange() Function
2. Using numpy.linspace() Function
3. Using numpy.logspace() Function

1. Using numpy.arange() Function

The numpy.arange() function creates an array by generating a sequence of numbers based on specified start, stop, and step values. It is similar to Python's built-in range() function but returns a NumPy array.

Syntax:

```
numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)
```

where,

- start (optional) − The starting value of the interval. Default is 0.
- stop − The end value of the interval (not included).
- step (optional) − The spacing between values. Default is 1.
- dtype (optional) − The desired data type for the array. If not given, NumPy interprets the data type from the input values.

Example	Output
import numpy as np ***# Create an array from 0 to 9*** arr = np.arange(10) print("Array using arange():", arr)	Array using arange(): [0 1 2 3 4 5 6 7 8 9]
import numpy as np ***# Create an array from 1 to 9 with a step of 2*** arr = np.arange(1, 10, 2) print("Array with start, stop, and step:", arr)	Array with start, stop, and step: [1 3 5 7 9]

2. Using numpy.linspace() Function

The numpy.linspace() function generates an array with evenly spaced values over a specified interval. It is useful when you need a specific number of points between two values. This function is similar to the arange() function. In this function, instead of step size, the number of evenly spaced values between the intervals is specified.

Syntax:

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

where,

- start − The starting value of the interval.
- stop − The end value of the interval.
- num (optional) − The number of evenly spaced samples to generate. Default is 50.
- endpoint (optional) − If True, stop is the last sample. If False, it is not included. Default is True.
- retstep (optional) − If True, returns (samples, step), where step is the spacing between samples. Default is False.
- dtype (optional) − The desired data type for the array. If not given, NumPy interprets the data type from the input values.
- axis (optional) − The axis in the result along which the samples are stored. Default is 0.

Example

```
import numpy as np
# Create an array of 10 evenly spaced values from 0 to 1
arr = np.linspace(0, 1, 10)
print("Array using linspace():", arr)
```

Output

```
Array using linspace(): [0.         0.11111111 0.22222222 0.33333333
0.44444444 0.55555556 0.66666667 0.77777778 0.88888889 1. ]
```

3. Using numpy.logspace() Function

The numpy.logspace() function generates an array with values that are evenly spaced on a log scale. This is useful for generating values that span several orders of magnitude.

Syntax:

numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)

where,

- start − The starting value of the sequence (as a power of base).
- stop − The end value of the sequence (as a power of base).
- num (optional) − The number of samples to generate. Default is 50.
- endpoint (optional) − If True, stop is the last sample. If False, it is not included. Default is True.
- base (optional) − The base of the log space. Default is 10.0.
- dtype (optional) − The desired data type for the array. If not given, NumPy interprets the data type from the input values.
- axis (optional) − The axis in the result along which the samples are stored. Default is 0.

Example

```
import numpy as np
# Create an array
arr = np.logspace(1, 10, 10, base=2)
print("Array with base 2:", arr)
```

Output

```
Array with base 2: [   2.    4.    8.   16.   32.   64.  128.  256.  512. 1024.]
```

2.2.5 Array Operations

NumPy supports a variety of array operations that allow you to perform efficient computations on arrays. These operations can be basically categorized into element-wise operations and matrix operations.

1. Element-wise Operations

These operations apply to each element of the array individually. This operation performs addition, subtraction, multiplication, division, etc., directly on arrays.

Example

```
import numpy as np

# Create two arrays

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

# Element-wise addition

c = a + b

# Element-wise multiplication

d = a * b

# Element-wise subtraction

e = a - b

# Element-wise division

f = a / b
```

Output

```
[5, 7, 9]          #addition

[4, 10, 18]        #multiplication

[-3, -3, -3]       #subtraction

[0.25, 0.4, 0.5]   #division
```

2. Matrix Operations

NumPy supports various matrix operations like matrix multiplication, dot product, etc. The **np.dot()** is used for matrix multiplication.

Example

```
# 2D arrays (matrices)
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
# Matrix multiplication (dot product)
c = np.dot(a, b)
print(c)
```

Output

```
[[19 22]
[43 50]]
```

2.2.5.1 Indexing

Array Indexing in NumPy is used to access or modify specific elements of an array. It allows retrieval of data from arrays by specifying the positions (indices) of elements. This is the important feature in NumPy that enables efficient data manipulation and analysis for numerical computing tasks.

1. Accessing 1D Array: In a 1D array, we can access elements using their zero-based index.

Example

```
import numpy as np
# Create a 1D NumPy array with five elements
arr = np.array([10, 20, 30, 40, 50])
# Access and print the first element of the array
print(arr[0])
```

Output

```
10
```

2. Access 2-D Array: To access elements, you specify the row index first and the column index second.

Example

```
import numpy as np
# Define a 2D array
```

```
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Access the element at row 1, column 2
print(matrix[1, 2])
```

Output

6

3. Accessing 3D Arrays: A 3D array can be visualized as a stack of 2D arrays. This need three indices:

- **Depth:** Specifies the 2D slice.
- **Row:** Specifies the row within the slice.
- **Column:** Specifies the column within the row.

Example

```
import numpy as np
cube = np.array([[[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]],
                 [[10, 11, 12],
                  [13, 14, 15],
                  [16, 17, 18]]])
# Access the element at depth 1, row 2, column 0
print(cube[1, 2, 0])
```

Output

16

2.2.5.2 Bitwise AND operator

In NumPy, the symbol & is used as a **bitwise AND operator** for arrays. However, it's also commonly used for **element-wise logical AND** operations when working with boolean arrays.

Example

```
import numpy as np
a = np.array([True, False, True])
```

```
b = np.array([True, True, False])

result = a & b

print(result)
```

Output

```
[ True False False]
```

2.2.5.3 Slicing

Slicing in NumPy works similarly to Python list slicing but is **more powerful** due to multi-dimensional support. The syntax to slicing in NumPy:

Syntax:

array[start:stop:step]

where,

- **start:** Index to begin slicing (inclusive)
- **stop:** Index to end slicing (exclusive)
- **step:** Step size (optional)

Example

```
import numpy as np

a = np.array([10, 20, 30, 40, 50])

print(a[1 : 4])

print(a[ : 3])

print(a[ : : 2])
```

Output

```
[20 30 40]

[10 20 30]

[10 30 50]
```

2.2.5.4 Joining of Arrays

Joining arrays in NumPy means combining multiple arrays into one. NumPy offers several functions to join arrays, depending on how you want to join them (horizontally, vertically, along a new axis, etc.). NumPy provides various functions to combine arrays. In this section, we will discuss some of the major ones.

- numpy.concatenate()

- numpy.stack()
- numpy.block()

1. Using numpy.concatenate(): The concatenate function in NumPy joins two or more arrays along a specified axis.

Syntax:

numpy.concatenate((array1, array2, …), axis=0)

The first argument is a tuple of arrays we intend to join and the second argument is the axis along which we need to join these arrays.

Example

```
import numpy as np
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])
res = np.concatenate((arr1, arr2))
print(res)
```

Output

```
[1 2 3 4]
```

2. **Using numpy.stack()**: The stack() function of NumPy joins two or more arrays along a new axis.

Syntax:

numpy.stack(arrays, axis=0)

Example

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])
res = np.stack((arr1, arr2), axis=1)
print(res)
```

Output

```
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
```

3. numpy.block(): numpy.block is used to create nd-arrays from nested blocks of lists.

Syntax:

numpy.block(arrays)

Example

```
import numpy as np
b1 = np.array([[1, 1], [1, 1]])
b2 = np.array([[2, 2, 2], [2, 2, 2]])
b3 = np.array([[3, 3], [3, 3], [3, 3]])
b4 = np.array([[4, 4, 4], [4, 4, 4], [4, 4, 4]])
block_new = np.block([
   [b1, b2],
   [b3, b4]])
print(block_new)
```

Output

```
[[1 1 2 2 2]
 [1 1 2 2 2]
 [3 3 4 4 4]
 [3 3 4 4 4]
 [3 3 4 4 4]]
```


2.2.5.5 Splitting

Splitting in NumPy refers to **dividing an array into multiple sub-arrays**. It is useful when you want to break down large datasets or process chunks of data separately.

Table 2.2.30: Example of split ()

FUNCTION	DESCRIPTION
np.split()	Split into equal parts
np.array_split()	Split into unequal parts if needed
np.hsplit()	Horizontal split (columns)
np.vsplit()	Vertical split (rows)
np.dsplit()	Depth split (for 3D arrays)

1. np.split(): Splits an array into equal parts along a specified axis.

Syntax:

np.split(array, sections, axis=0)

where

- array: The input array.
- sections: Number of equal parts to split into.
- axis: Axis along which to split (default is 0).

Example

```
import numpy as np
a = np.array([10, 20, 30, 40, 50, 60])
result = np.split(a, 3)
print(result)
```

Output

```
[array([10, 20]), array([30, 40]), array([50, 60])]
```

2. **np.array_split():** Like split(), but allows unequal splits if the array length isn't divisible evenly

Syntax:

np.array_split(array, sections, axis=0)

Example

```
b = np.array([1, 2, 3, 4, 5])
result = np.array_split(b, 3)
print(result)
```

Output

```
[array([1, 2]), array([3, 4]), array([5])]
```

3. **np.hsplit():** Splits an array horizontally (column-wise) for 2D arrays.

Syntax:

np.hsplit(array, sections)

Example

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])
```

print(np.hsplit(a, 3))

Output

[array([[1], [4]]), array([[2], [5]]), array([[3], [6]])]

4. **np.vsplit():** Splits an array vertically (row-wise).

Syntax:

np.vsplit(array, sections)

Example

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])
print(np.vsplit(a, 2))
```

Output

[array([[1, 2, 3]]), array([[4, 5, 6]])]

5. **np.dsplit()**: Splits an array along depth (3rd axis) for 3D arrays.

Syntax:

np.dsplit(array, sections)

Example

```
a = np.array([[[1, 2], [3, 4]],
              [[5, 6], [7, 8]]])
print(np.dsplit(a, 2))
```

Output

[array([[[1], [3]], [[5], [7]]]), array([[[2], [4]], [[6], [8]]])]

2.2.6 Familiarization of Pandas, Matplotlib, SciPy and Scikit-Learn

Python has become one of the most popular programming languages for data analysis, scientific computing, and machine learning, largely due to its powerful ecosystem of libraries. Among these **Pandas, Matplotlib, SciPy and Scikit-Learn** stand out as essential tools for handling different aspects of the data science workflow. Each library serves a unique purpose, Pandas is used for data manipulation and analysis, Matplotlib for data visualization, SciPy for scientific and technical computations, and Scikit-learn for building machine learning models.

Familiarizing yourself with these libraries enables you to work more effectively with data from cleaning and organizing it, to exploring trends through visualizations, to applying advanced algorithms for predictive modeling. Together, these form a comprehensive toolkit that supports end-to-end data science projects and makes Python a go-to language for professionals in fields ranging from finance and healthcare to engineering and research.

2.2.6.1 Pandas

The pandas library is a powerful Python library used for data manipulation and analysis. It provides high-level data structures and functions designed to make working with structured or tabular data fast, easy, and expressive.

Key features

1. **DataFrame:** The DataFrame is a two-dimensional labeled data structure with columns of potentially different types. It's similar to a spreadsheet or SQL table, and it is the primary data structure used in pandas.

2. **Series:** A one-dimensional labeled array capable of holding any data type.

3. **Data Alignment:** The library automatically aligns data based on label, simplifying operations like joining and merging data.

4. **Data Manipulation:** pandas provides functionalities for reshaping, slicing, indexing, and merging datasets.

5. **Data I/O:** It supports reading and writing data from and to various file formats such as CSV, Excel, SQL databases, and more.

6. **Time Series Analysis:** pandas has robust support for working with time series data.

7. **Data Cleaning:** It offers various methods for handling missing data, removing duplicates, and transforming data.

8. **GroupBy:** pandas allows for splitting, applying functions, and combining data based on some criteria.

2.2.6.2 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is widely used for generating plots, charts, histograms, and other graphical representations of data.

Key features

1. **Wide Range of Plotting Functions**: Matplotlib provides a plethora of functions for creating various types of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and more.

2. **Customization:** The library allows for extensive customization of plot

appearance, including colors, line styles, markers, fonts, labels, and annotations.

3. **Multiple Output Formats:** Matplotlib supports multiple output formats, including PNG, PDF, SVG, and interactive formats for use in Jupyter Notebooks or web applications.

4. **Integration with NumPy:** Matplotlib seamlessly integrates with NumPy, making it easy to plot data stored in NumPy arrays or generated using NumPy functions.

5. **Multiple APIs:** Matplotlib provides two main APIs for creating plots: a MATLAB-style state- based interface (pyplot) and an object-oriented interface. The pyplot interface is convenient for quick plotting tasks, while the object-oriented interface offers more control and flexibility for complex plotting scenarios.

6. **Backend Agnostic:** Matplotlib supports different backends for rendering plots, allowing you to generate plots in various environments, including desktop applications, web applications, and batch scripts.

7. **Integration with Pandas:** Matplotlib works seamlessly with pandas DataFrames, allowing you to create plots directly from DataFrame objects.

2.2.6.3 SciPy

SciPy is a powerful open-source library in Python used for scientific and technical computing. It builds on top of NumPy, providing additional functionality for optimization, integration, interpolation, linear algebra, statistics, signal processing, and more.

Key features

1. **Numerical Integration:** SciPy offers functions for numerical integration, including single and multiple integration, numerical quadrature, and numerical solutions to ordinary differential equations (ODEs).

2. **Optimization:** It provides optimization routines for unconstrained and constrained optimization, nonlinear least-squares fitting, and global optimization.

3. **Interpolation:** SciPy includes interpolation functions for one-dimensional and multi- dimensional data, including spline interpolation and radial basis function interpolation.

4. **Linear Algebra:** It extends the linear algebra capabilities of NumPy with additional functions for solving linear equations, eigenvalue problems, matrix factorizations, and sparse matrix operations.

5. **Statistics:** SciPy offers statistical functions for probability distributions,

statistical tests, descriptive statistics, and random number generation.

6. **Signal Processing:** It provides a wide range of signal processing functions, including filtering, Fourier analysis, wavelet transforms, and signal generation.

7. **Image Processing:** SciPy includes functions for image manipulation, filtering, and analysis.

8. **Sparse Matrices:** It provides support for sparse matrices and includes functions for sparse linear algebra and sparse matrix operations.

2.2.6.4 Scikit-Learn

Scikit-learn, often abbreviated as sklearn, is a popular Python library for machine learning tasks such as classification, regression, clustering, dimensionality reduction, and model selection. It is built on top of NumPy, SciPy, and matplotlib, and it provides a simple and efficient interface for implementing various machine learning algorithms.

Key features

1. **Consistent Interface:** Scikit-learn provides a unified interface across different algorithms, making it easy to experiment with various models without needing to learn new APIs for each algorithm.

2. **Wide Range of Algorithms:** The library includes a vast collection of supervised and unsupervised learning algorithms, including linear models, support vector machines, decision trees, random forests, k-nearest neighbors, clustering algorithms, dimensionality reduction techniques, and more.

3. **Model Evaluation and Validation:** Scikit-learn offers tools for model evaluation, cross- validation, and hyperparameter tuning to help users select the best model for their data and avoid overfitting.

4. **Feature Extraction and Transformation:** It provides utilities for feature extraction from raw data, feature scaling, feature selection, and transformation pipelines.

5. **Integration with NumPy and SciPy:** Scikit-learn seamlessly integrates with NumPy and SciPy, allowing users to work with data stored in NumPy arrays and take advantage of the scientific computing capabilities of SciPy.

6. **Ease of Use:** The library is designed with simplicity and ease of use in mind, making it accessible to both beginners and experienced machine learning practitioners.

7. **Extensibility:** Scikit-learn is an open-source project with an active community of contributors, and it is designed to be easily extensible, allowing users to implement custom algorithms or extend existing ones.

Recap

- Python libraries are tools that allow developers to **access reusable code** for specific tasks like data processing, visualization, or machine learning.
- Libraries work by providing **pre-defined functions, classes, and modules** that can be imported and used in a Python script.
- **NumPy** is essential for handling **large numerical datasets** and performing fast mathematical computations.
- The **ndarray** object in NumPy is a powerful N-dimensional array structure that forms the basis of all computations.
- NumPy arrays can hold only **one data type**, which makes them more memory-efficient than regular Python lists.
- NumPy creates **1D arrays** (like a list of numbers) and **2D arrays** (like a matrix) using the np.array() function.
- **Array attributes** such as .shape, .ndim, and .dtype give important information about the structure and type of data in the array.
- NumPy supports **element-wise operations** such as addition and multiplication, and also **matrix operations** like dot product.
- **Data types** in NumPy include integers, floats, booleans, and complex numbers, and they help in optimizing performance.
- **Indexing** allows you to access individual elements in NumPy arrays using position (e.g., arr[0]).
- **Slicing** is used to retrieve a range of elements from arrays or DataFrames using the start:stop format (e.g., arr[1:4]).
- **Joining** combines arrays or data tables, using functions like np.concatenate() in NumPy.
- **Splitting** divides arrays or datasets into parts using np.split() for arrays.
- **Pandas** is used for loading, manipulating, and analyzing structured data in the form of tables.
- In Pandas, a **Series** is a one-dimensional labeled array, while a **DataFrame** is a two-dimensional labeled data structure similar to an Excel sheet.
- **Matplotlib** allows users to create a wide range of **static, animated, and interactive plots** to visualize data trends and patterns.
- **SciPy** extends NumPy's functionality by adding modules for advanced **scientific and technical computations**.

- SciPy includes modules like optimize, integrate, and stats, which are used for mathematical operations and data analysis.
- **Scikit-learn** offers a wide variety of machine learning models and tools for **data preprocessing, model training, testing, and evaluation**.

Objective Type Questions

1. Which Python keyword is used to include a library?
2. Which library is mainly used for numerical computations?
3. What is the primary array type in NumPy?
4. Which attribute shows the dimensions of a NumPy array?
5. Which attribute returns the number of array dimensions?
6. Which data type represents decimal values in NumPy?
7. What is the process of accessing a single element called?
8. What is used to access a range of values in arrays?
9. What operation is used to combine multiple arrays?
10. Which operation divides an array into parts?
11. What is a one-dimensional labeled data structure in Pandas?
12. Which Pandas object is two-dimensional and table-like?
13. Which library is used to plot graphs and charts in Python?
14. Which library provides tools for scientific calculations?
15. Which library is widely used for machine learning in Python?

Answers to Objective Type Questions

1. Import
2. NumPy
3. Ndarray
4. Shape
5. Ndim
6. Float
7. Indexing
8. Slicing
9. Joining
10. Splitting
11. Series
12. DataFrame
13. Matplotlib
14. SciPy
15. Scikit-learn

Assignments

1. What is a Python library? Give two examples. Why are libraries important in Python programming? What is NumPy used for in Python?
2. Define an array in NumPy. Describe one dimensional, two dimensional and N-dimensional array in NumPy. How can you create an array using existing data in NumPy?
3. Briefly define array attributes with examples. Explain the role of array operations in NumPy.
4. Write a detailed note on Pandas, Matplotlib, SciPy, Scikit-learn libraries in Python?

References

1. VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data*. O'Reilly Media.

2. McKinney, W. (2018). *Python for data analysis: Data wrangling with pandas, NumPy, and IPython* (2nd ed.). O'Reilly Media.

3. Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. *Computing in Science & Engineering*, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55.

4. Raschka, S., and Mirjalili, V. (2019). *Python machine learning* (3rd ed.). Packt Publishing.

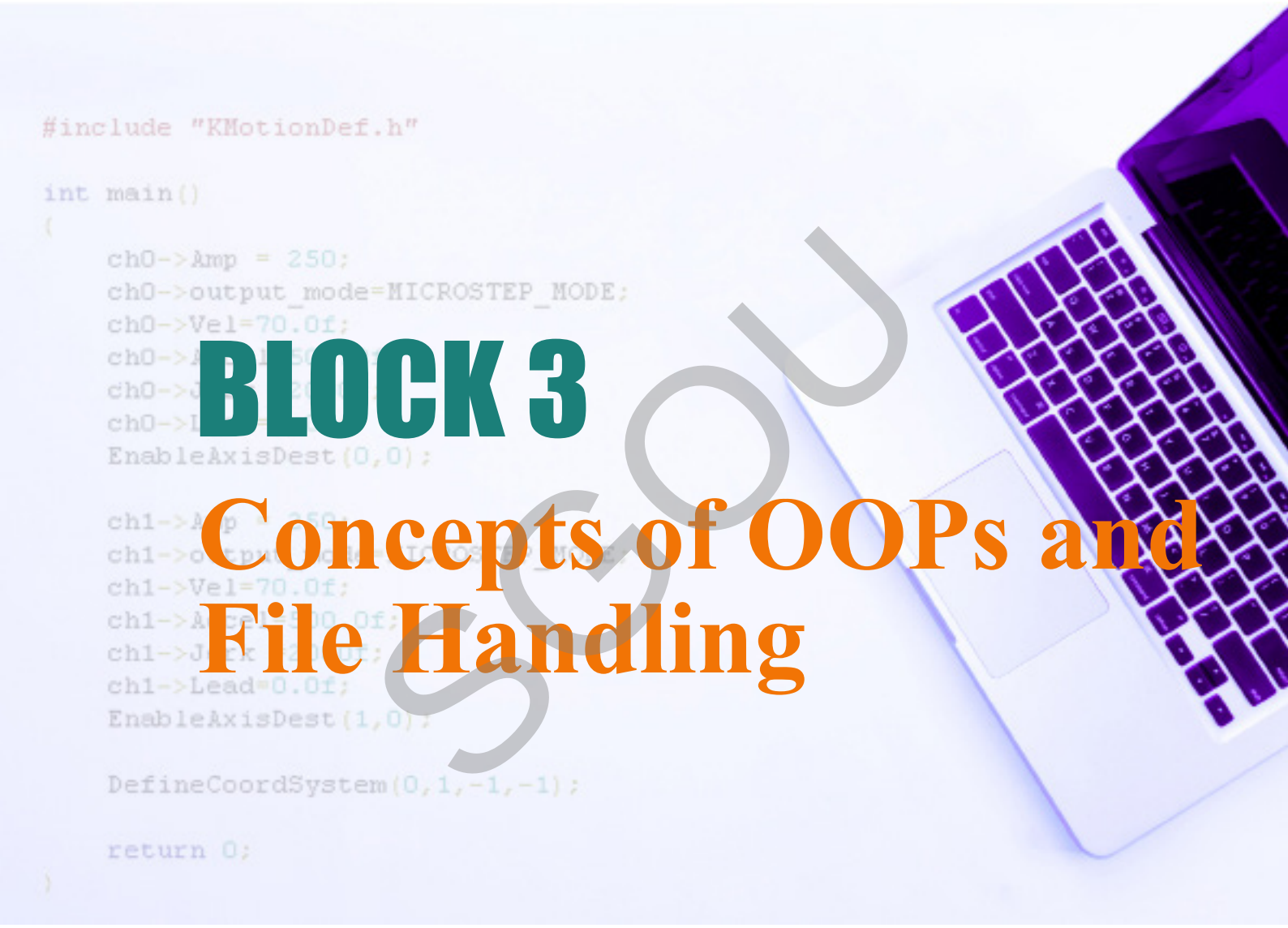
Suggested Reading

1. W3Schools. (n.d.). *NumPy tutorial*. Retrieved May 19, 2025, from https://www.w3schools.com/python/numpy/default.asp

2. GeeksforGeeks. (n.d.). *Python programming language*. Retrieved May 19, 2025, from https://www.geeksforgeeks.org/python-programming-language/

3. Real Python. (n.d.). *Python tutorials*. Retrieved May 19, 2025, from https://realpython.com/

4. NumPy Developers. (n.d.). *NumPy documentation*. Retrieved May 19, 2025, from https://numpy.org/doc/stable/



BLOCK 3

Concepts of OOPs and File Handling

Concepts of Object Oriented Programming (OOP)

Learning Outcomes

At the end of this unit, the learner will be able to:

- define the term class in Python.
- list the basic features of object-oriented programming in Python such as class, object, encapsulation, and inheritance.
- recall the syntax for creating a class and an object in Python.
- identify the types of inheritance supported in Python.
- familarise the purpose of the __init__() method in a Python class

Prerequisites

Before learning Object-Oriented Programming in Python, you may already be familiar with writing simple Python programs using variables, functions, loops, and conditional statements. You might have written programs that calculate the sum of numbers, display messages, or process user input. These are examples of procedural programming, where code is written step by step to perform tasks.

Now, imagine you are working on a large project such as a library system, an online store, or a student management application. In such cases, using just functions and variables may not be enough to organize your code well. This is where Object-Oriented Programming (OOP) becomes useful.

OOP helps us to group related data and functions together using structures called classes and objects. You may already know how real-life objects (like a car or a mobile phone) have properties (like color or brand) and functions (like start or call). In the same way, objects in programming have attributes and behaviors.

In this unit, you will learn how to create and use classes and objects, how to protect data using encapsulation, and how to share features using inheritance. These ideas will help you write code that is more organized, reusable, and easier to understand.

Key Concepts

class, object, encapsulation, inheritance, polymorphism

Discussion

3.1.1 Introduction

In Python, Object-Oriented Programming (OOP) is a way of writing code that models real-world things using objects and classes. It helps organize code by bundling data (like a student's name or marks) and the functions that work on that data together. Key ideas in OOP include encapsulation (hiding internal details), inheritance (reusing code from one class in another), and polymorphism (using the same function name for different types of actions). OOP makes code easier to understand, maintain, and expand.

3.1.2 Concepts of Object-Oriented Programming (OOP)

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

3.1.2.1 Class

A class is like a blueprint or template used to create objects. It defines the attributes (variables) and methods (functions) that the objects will have.

To understand why we need classes, imagine you're keeping track of dogs. Each dog has properties like breed and age. If you try to manage this using simple lists, it becomes confusing, especially with hundreds of dogs and many characteristics. You wouldn't know which value refers to which property. This can lead to messy and disorganized code.

A class helps organize this data by grouping related properties and behaviors together. For example, a Dog class can include attributes like breed and age, and methods like bark() or sleep().

In Python, we define a class using the class keyword. The variables inside a class are called attributes, and they can be accessed using the dot (.) operator.

Syntax of Class Definition:

class ClassName:

```
# Statement 1
#  Statement 2
 .
 .
 .
# Statement N
```

Example: Creating an Empty Class

```
class Dog:
  pass
```

3.1.2.2 Objects

In Python, an object is an instance of a class and represents a real-world entity. It contains three important characteristics: **state**, **behavior**, and **identity**. The state of an object refers to its properties or attributes, such as the color or age of a dog. The behavior refers to the actions or methods that the object can perform, like barking or sleeping in the case of a dog. Identity is what makes the object unique and helps distinguish it from other objects, the dog's name. In everyday terms, objects can be anything around us, like a chair, a pen, or a mobile phone.

In Python, even values like strings, numbers, and lists are also treated as objects.

- "Hello" is a string object.
- 12 is an integer object.
- [1, 2, 3] is a list object.

To better understand the concepts of state, behavior, and identity in object-oriented programming, let us consider a simple example using a Dog class. The **identity** of the dog can be represented by its name, which makes it unique from other dogs. The **state** includes attributes like the dog's breed, age, and color, which describe its characteristics. The **behavior** refers to actions the dog can perform, such as eating or sleeping, which are defined as methods in the class. For example, we can create an object of the Dog class by writing obj = Dog(). This statement creates an object named obj that belongs to the Dog class. Once created, this object can access the attributes and behaviors defined in the class. Let's first grasp the fundamental terms that will be while working with objects and classes before delving further into them.

1.self

In Python, every method defined inside a class must have at least one parameter, and by convention, it is named self. This self parameter refers to the current instance of the class and allows access to its attributes and methods. When you call a method using an

object, you don't need to pass anything for self; Python handles that automatically. For example, if you write myobject.method(5), Python internally translates it to ClassName. method(myobject, 5). This behavior is similar to how this works in Java or this pointer in C++. The self keyword is what makes it possible for each object to keep track of its own data.

2. init method

The __init__ method in Python is a special method used to initialize a new object of a class. It is similar to constructors in languages like Java or C++. The name __init__ stands for "initialize." When you create a new object, Python automatically calls the __init__ method to set up the object with initial values.

Syntax

```
class ClassName:

    def __init__(self, parameters):
```

Example:

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
s1 = Student("Alice", 20)

print("Name:", s1.name)
print("Age:", s1.age)
```

Output:

```
Name: Alice
Age: 20
```

Example 1: Creating a Class and Object with Class and Instance Attributes

```
class Dog:
    attr1 = "mammal"
    def __init__(self, name):
        self.name = name
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")
print("Rodger is a", Rodger.__class__.attr1)
print("Tommy is also a", Tommy.__class__.attr1)
print("My name is", Rodger.name)
print("My name is", Tommy.name)
```

Output:

```
Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy
```

Example 2: Creating a Class and Objects with Methods

```
class Dog:
    attr1 = "mammal"
     def __init__(self, name):
        self.name = name
    def speak(self):
        print("My name is", self.name)
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")
Rodger.speak()
Tommy.speak()
```

Output:

```
My name is Rodger
My name is Tommy
```

3.1.2.3 Inheritance

Inheritance is a key concept in object-oriented programming. It allows one class (called the child class or derived class) to inherit or receive the features (like variables and methods) of another class (called the parent class or base class). This means that the child class can use the code written in the parent class without rewriting it. Inheritance helps in code reusability, making programs shorter and easier to maintain. It also reflects real-world relationships. For example, if we have a class called Animal, we can create a child class Dog that inherits from Animal, meaning Dog will have all the features of Animal along with its own specific features. Inheritance can also be transitive, meaning if class B inherits from class A, and class C inherits from class B, then class C also inherits the features of class A.

Types of inheritance

In Python, there are five main types of inheritance, each defining a different way in which classes relate to one another.

1. Single Inheritance

Single Inheritance is a type of inheritance in which a child class inherits from only one parent class. This means the child class gets all the properties and behaviors (methods and attributes) of that single parent class. As shown in Figure 3.1.1, class B inherits from class A, which means that all the attributes and methods defined in class A become directly available to class B without redefining them.

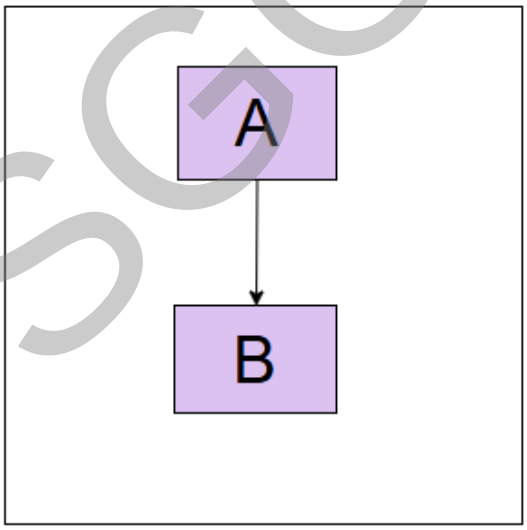


Fig. 3.1.1 Single Inheritance

2. Multiple Inheritance

Multiple Inheritance is a type of inheritance a child class inherits from more than one parent class. This means the child class can access the properties and behaviors of all the parent classes. As shown in Figure 3.1.2, class C inherits from both class A and class B, so it can use all the methods and attributes from both classes.

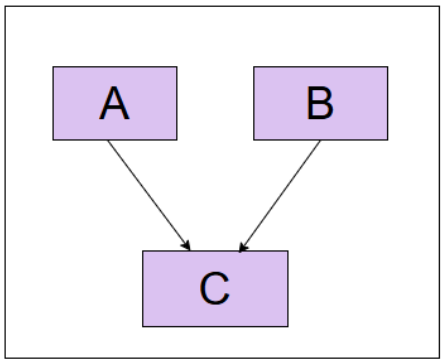


Fig. 3.1.2 Multiple Inheritance

3. Multilevel Inheritance

Multilevel Inheritance is a type of inheritance in which a class is derived from a child class, which in turn is derived from another parent class. In this hierarchy, each level inherits properties and methods from its immediate parent. As shown in Figure 3.1.3, class B inherits from class A, and then class C inherits from class B. This means class C gets all the properties and behaviors of class B, as well as those of class A. The inheritance flows through the levels, allowing class C to access everything from both B and A.

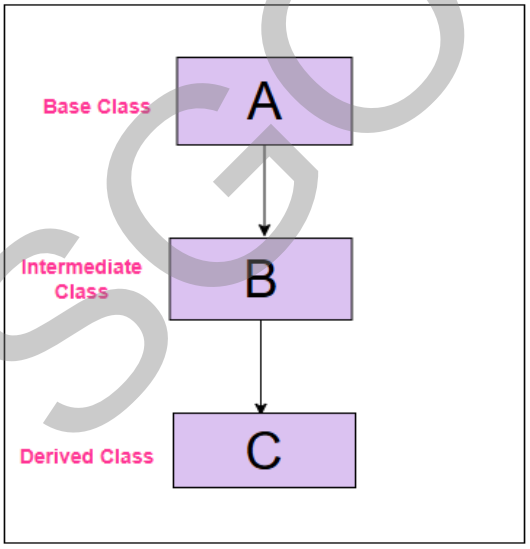


Fig. 3.1.3 Multilevel Inheritance

4. Hierarchical Inheritance

Hierarchical Inheritance in object-oriented programming refers to a situation where multiple child classes inherit from a single parent class. This means that the same base class serves as a common source of properties and methods for several subclasses. As illustrated in Figure 3.1.4, class B, class C, and class D all inherit from a single parent class A. This implies that B, C, and D can each use the methods and attributes defined in A, but they are independent of one another. Such a structure promotes code reusability and allows multiple subclasses to maintain consistency by sharing a common base class.

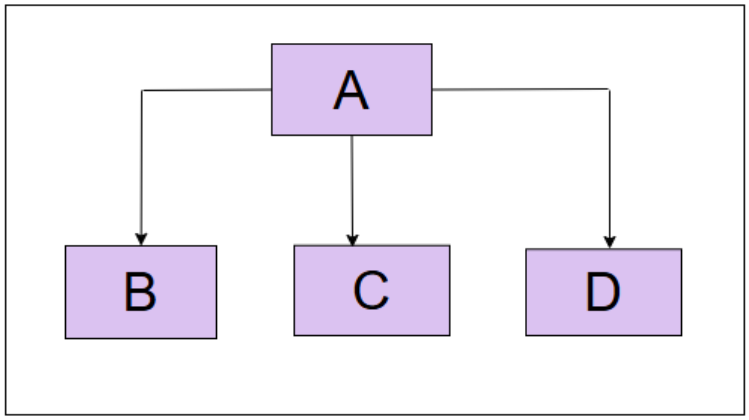


Fig. 3.1.4 Hierarchical Inheritance

3.1.2.4 Polymorphism

Polymorphism in Python is a concept where one function or method can behave differently depending on the object it is acting upon. The word "polymorphism" comes from Greek, meaning "many forms." In Python, it means that a single function or method can work with different types of objects, giving different results for each object.

Imagine you have different animals, and they all make sounds. You can create a method called make_sound() that, but each animal makes its own unique sound. So, the same method name (make_sound) will behave differently depending on which animal object is calling it.

Example:

```
class Dog:
    def make_sound(self):
        print("Bark")
class Cat:
    def make_sound(self):
        print("Meow")
dog = Dog()
cat = Cat()
dog.make_sound()
cat.make_sound()
```

Output:

Bark

Meow

Types of Polymorphism

Polymorphism in object-oriented programming (OOP) can be classified into two main types**:**

1. Compile-time Polymorphism
2. Runtime Polymorphism

1. Compile-time Polymorphism (Static Polymorphism)

Compile-time polymorphism happens when the method to be called is determined at compile time, before the program is run. This type of polymorphism is achieved using method overloading or operator overloading.

Method overloading is when multiple methods have the same name but differ in the number or type of their parameters. The correct method is chosen based on the arguments passed when calling the method.

Example:

Program:

```
class MathOperations:
    def add(self, a, b):
        return a + b
    def add(self, a, b, c):
        return a + b + c
math = MathOperations()
print(math.add(5, 10, 15))
```

Output:

30

2. Runtime Polymorphism (Dynamic Polymorphism)

Runtime polymorphism happens when the method to be called is determined at runtime, during the execution of the program. This type of polymorphism is achieved using method overriding.

Method overriding occurs when a subclass provides its own implementation of a method that is already defined in the parent class. The version of the method that is called depends on the object that is used to invoke it.

Example:

Program:

```
class MathOperations:
    def calculate(self, a, b):
        return a + b
class AdvancedMath(MathOperations):
    def calculate(self, a, b):
        return a * b
basic = MathOperations()
advanced = AdvancedMath()
print("Basic addition:", basic.calculate(5, 3))
print("Advanced multiplication:", advanced.calculate(5, 3))
```

Output:

```
8
15
```

3.1.2.5 Encapsulation

Encapsulation is one of the fundamental principles of object-oriented programming (OOP). It refers to the concept of hiding the internal details of how an object works and only exposing a controlled interface to the outside world. In Python, this is achieved by wrapping data (variables) and methods (functions) into a single unit, usually a class, and restricting direct access to some of the object's components. This protects the object's integrity by preventing unintended interference and misuse.

Encapsulation allows programmers to define access levels for class members using public, protected, or private access modifiers. By doing so, sensitive data can be kept hidden from direct access and only modified through well-defined interfaces like getter and setter methods. This not only enhances data security but also makes the code more modular, maintainable, and easier to debug.

1. **Public Members**

Public members are class attributes (variables) or methods (functions) that can be accessed from anywhere, both inside and outside the class. In Python, by default, all members of a class are public unless explicitly specified otherwise.

Example

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
s = Student("Helen", 23)
print("Name:", s.name)
print("Age:", s.age)
```

Output:

Name: Helen

Age: 23

In the above example, name and age are public members of the Student class. They can be accessed and modified directly using the object s.

2. Protected Members

Protected members are variables or methods that can be accessed within the class and its subclasses, but should not be accessed directly from outside the class. In Python, protected members are defined by prefixing the name with a single underscore (e.g., _name).

Example

```
class Student:
    def __init__(self, name, age):
        self._name = name
        self._age = age
s = Student("Bob", 21)
print("Name:", s._name)
print("Age:", s._age)
```

Output:

Name: Bob

Age: 21

3. Private Members

Private members are variables or methods in a class that cannot be accessed directly from outside the class. In Python, we make something private by putting two underscores (__) in front of its name.

Example

```
class Student:
    def __init__(self, name, marks):
        self.__name = name
        self.__marks = marks
    def display(self):
        print("Name:", self.__name)
        print("Marks:", self.__marks)
s = Student("Helen", 28)
s.display()
```

Output:

Name: Helen

Marks: 28

Recap

Class

- A blueprint for creating objects.
- Contains attributes (variables) and methods (functions).
- Syntax uses class keyword.

Object

- Instance of a class representing a real-world entity.
- Has state (attributes), behavior (methods), and identity (uniqueness).
- Created using syntax like obj = ClassName().

Inheritance

- Allows one class to reuse code from another class
- Promotes code reuse and reflects real-world hierarchies.

Types of Inheritance in Python:

- **Single Inheritance**: One child inherits from one parent.
- **Multiple Inheritance**: Child class inherits from more than one parent.
- Multilevel Inheritance: Chain of inheritance (A → B → C).
- **Hierarchical Inheritance**: Multiple children inherit from one parent.

Polymorphism

- Allows the same method name to perform different tasks based on the object.
- Example: make_sound() behaves differently in Dog and Cat.

Types of Polymorphism:

- **Compile-time (Static) Polymorphism**
  - Achieved via method overloading or operator overloading.
  - Python does not support true method overloading natively.
  - Latest defined method is used (e.g., multiple add() methods).
- **Runtime (Dynamic) Polymorphism**
  - Achieved through method overriding in subclasses.

Encapsulation

- Bundles data (attributes) and methods within a class.
- Protects data from outside interference and misuse.
- Achieved using private variables (prefix with _ or __).
- Promotes data hiding and modularity.

Objective Type Questions

1. What is the term used to describe a blueprint for creating objects in Python?
2. What keyword is used to define a class in Python?
3. What is the purpose of the __init__() method in a class?
4. Which keyword refers to the current instance of the class?
5. What do we call the process of combining data and methods within a class?
6. What symbol is used to indicate a private attribute in Python?
7. Name the concept that allows a child class to access methods and attributes of a parent class.
8. What is the name of the concept where the same method behaves differently depending on the object?
9. What is the term for a class that inherits properties from another class?
10. Which OOP concept helps in hiding internal object details from the outside world?
11. Name the type of inheritance where a class is derived from a single base class.
12. What is the type of inheritance where one class is derived from two or more base classes?
13. What is the name of the inheritance type where a class inherits from a derived class, forming a chain?
14. What is the name of the inheritance where multiple classes inherit from the same parent class?

Answers to Objective Type Questions

1. Class
2. Class
3. To initialize the object's attributes when it is created
4. Self
5. Encapsulation
6. Underscore (_ or __)
7. Inheritance
8. Polymorphism
9. Derived class or Subclass
10. Encapsulation
11. Single Inheritance
12. Multiple Inheritance
13. Multilevel Inheritance
14. Hierarchical Inheritance

Assignments

1. Explain the four main principles of Object-Oriented Programming (OOP) in Python with suitable examples.
2. Define a class named Student with attributes name and age. Write a method to display the student's details. Create two student objects and display their details.
3. Create a class Rectangle with attributes length and breadth. Write methods to calculate area and perimeter. Create an object and display the results.
4. Create a class Calculator with methods to perform addition, subtraction, multiplication, and division. Take user input for operations and display the result.
5. Write a program to demonstrate polymorphism using a method named speak() in two different classes Cat and Dog, both having different implementations of the method.

References

1. https://nptel.ac.in/courses/106106145

Suggested Reading

1. Lutz, Mark. Learning *python: Powerful object-oriented programming*. " O'Reilly Media, Inc.", 2013.
2. Goldwasser, Michael H., and David Letscher. *Object-Oriented Programming in Python*. Pearson Prentice Hall, 2008.
3. Zelle, John M. *Python programming: an introduction to computer science*. Franklin, Beedle & Associates, Inc., 2004.
4. B Downey, Allen. "Think Python: How to Think Like a Computer Scientist-2e." (2012).
5. Chun, Wesley J. Core *Python Applications Programming*. Pearson Education India, 2012.

UNIT 2 File Handling

Learning Outcomes

The students will be able:

- narrate the need of file handling for permanent data storage and retrieval.
- make aware of opening and closing of text files using different modes with the open() and close() method, and why the with clause is useful.
- discuss the Cursor handling functions like seek() and tell().
- explain the use of seek() and tell() functions to move the cursor and determine its position within a file.

Prerequisites

Files provide essential functions in computing. Firstly, they enable data persistence by storing information on disk, ensuring it remains accessible even after the program that created it ends. This is crucial for saving user input, application settings, and other critical data. Secondly, files facilitate data sharing between programs and systems, allowing data written by one program to be read by another, promoting communication and interoperability. Additionally, files are vital for data analysis, enabling programs to process and analyze large datasets, conduct tasks like data mining, statistical analysis, and reporting. Furthermore, files serve as a means of data backup, safeguarding against data loss due to hardware failure or accidental deletion. Lastly, in configuration management, files store settings and preferences, allowing programs to customize and configure software at runtime for improved usability

Keywords

File Handling Functions, Cursor Handling, File modes, open(), close(), File pointers.

Discussion

3.2.1 File Handling in python

File handling in Python is a fundamental concept that allows to read from and write to files using built-in functions. Python programs typically accept input, process it, and display output during execution, but this output is temporary and input must be manually provided each time. This limitation exists because variables retain their values only while the program is running. To ensure data such as user inputs and generated results remain available for future use, a permanent data storage is essential. In practical applications, information like employee records, inventory details, and sales figures must be preserved to avoid redundant data entry. This is achieved by saving data on secondary storage devices, ensuring it can be reused whenever needed.

3.2.2 Types of files

Files are primarily classified into two types: text files and binary files. Text files contain human-readable characters and can be opened within any standard text editor. It stores information in the form of a stream of ASCII or Unicode characters. Binary files consist of non-readable characters and symbols that require specialized software to access their contents. It stores the information in the form of a stream of bytes.

3.2.3 Opening and Closing a text file

Before reading from or writing to a file, it is necessary to open it using Python's built-in `open()` function. This function requires specifying a mode that indicates the intended operation, such as reading, writing, or appending.

3.2.3.1 Opening a file

1. To open a file in Python, use the open() function.

The syntax of open() is as follows:

file_obj= open(file_name, mode)

The file_obj serves as a bridge between the program and the data file located in permanent storage. The attribute `<mode>` returns the mode in which a file was opened, while `<file_name>` provides the name of the file. The `file_name` refers to the name of the file that needs to be opened. If the file is not located in the current working directory, the full path along with the file name must be specified. The `mode` is an optional parameter that defines how the file should be accessed. This mode determines the operation to be performed on the file, such as `'r'` for reading, `'w'` for writing, `'+'` for both reading and writing, and `'a'` for appending data to the end of an existing file. By default, files are opened in read mode. The `open ()` function in Python is used to open a file so that

it can be read from or written to. It establishes a connection between the file stored on disk and the program.

Table: 3.2.1 File open modes

File mode	Description	File offset position
"r"	Opens the file in read-only mode.	Beginning of the file
"r+"	To both read from and write to the file. It will replace any prior data in the file.	Beginning of the file
"w"	Opens the file in write mode. If the file already exists, all the contents will be overwritten. If the file doesn't exist, then a new file will be created.	Beginning of the file
"w+"	To read and write data It will replace current data.	Beginning of the file
"a"	Opens the file in append mode. If the file doesn't exist, then a new file will be created.	End of the file
"a+"	Opens the file in append and read mode. If the file doesn't exist, then it will create a new file.	End of the file

Example: To open and read the content of a file "data.txt" – (First, create a text file named **data.txt** with some content, and then execute the code in the python terminal)

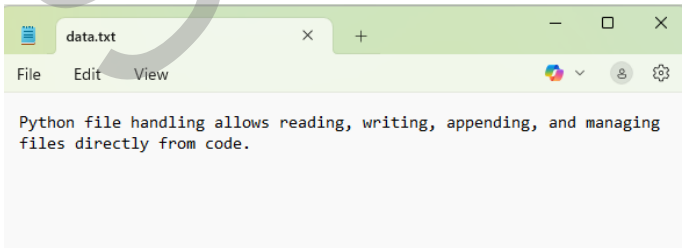


Fig: 3.2.2 data.txt

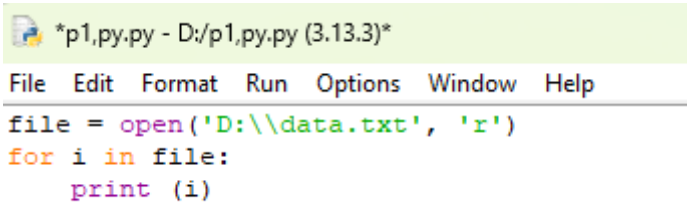


Fig: 3.2.3 Python code to open and read the contents of a text file using open() and for loop

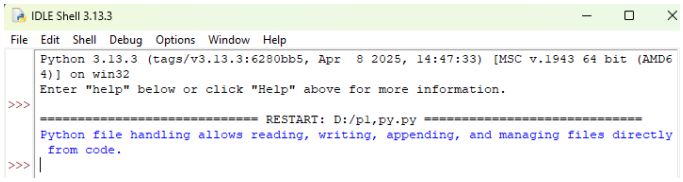


Fig: 3.2.4 Display the content of file.

The `open` function will open the file in read-only mode, and the `for` loop will print each line of the file.

2. Opening a file using with clause

In Python, we can also open a file using with clause.

The syntax of with clause is:

with open (file_name, access_mode) as file_ obj:

The advantage of using with clause is that any file that is opened using this clause is closed automatically. In this case, explicitly using the `close()` statement isn't required. Python handles the file closing automatically.

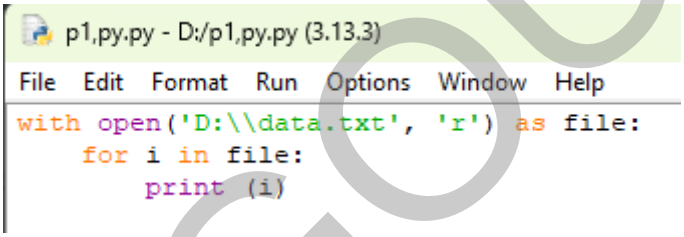


Fig: 3.2.5 Python code to open and read the contents of a text file using “with” clause.

3.2.3.2 Closing a file

After completing read or write actions on a file, closing it is recommended. Python offers the `close()` method for this purpose. When a file is closed, the system releases its allocated memory.

The syntax of close() is:

file_obj.close()

where `file_obj` is the variable assigned when the file was opened. It is generally good practice to close files after use. If a file object is reassigned to a different file, the original file will be closed automatically.

3.2.4 Creating a file with write () mode and append () mode

To write data into a file, it's necessary to first open it in either write mode or append mode. When an existing file is opened in write mode, any prior content is deleted, and the file pointer is set at the beginning. Conversely, append mode adds new data to the end of the existing content, as the file pointer is positioned at the file's end. We can use the following methods to write data in the file.

- write() - for writing a single string
- writelines() - for writing a sequence of strings

write() -

p2.py - C:/Users/USER 16/p2.py (3.13.3)

File Edit Format Run Options Window Help

```
# Opening a file in write mode ('w')
file = open("my_file.txt", "w")

# Writing some text to the file
file.write("This is the first line.\n")
file.write("This is the second line.\n")

# Closing the file
file.close()
# Opening a file in read mode ('r')then only read the data
file = open("my_file.txt", "r")
for i in file:
    print(i)
```

Fig: 3.2.6 File open in write mode, to see the content open the file in read mode.

IDLE Shell 3.13.3

File Edit Shell Debug Options Window Help

```
Python 3.13.3 (tags/v3.13.3:6280bb5, Apr  8 2025, 14:47:3
AMD64)] on win32
Enter "help" below or click "Help" above for more informa
>>>
========== RESTART: C:/Users/USER 16/p2.py =
This is the first line.

This is the second line.

>>>
```

Fig: 3.2.7 content of a file my_file.txt

Opening the file in append mode ('a') - The following program will position the "cursor" at the very end of the existing file. Any new data write will be added after the current content.

p2.py - C:/Users/USER 16/p2.py (3.13.3)

File Edit Format Run Options Window Help

```
# Opening the file in append mode ('a')
file = open("my_file.txt", "a")
# Appending new text to the end of the file
file.write("This line is appended.\n")
file.write("Another line appended.\n")
# Closing the file
file.close()
file = open("my_file.txt", "r")
for i in file:
    print(i)
```

Fig: 3.2.8 File open in append mode, to see the content open the file in read mode.

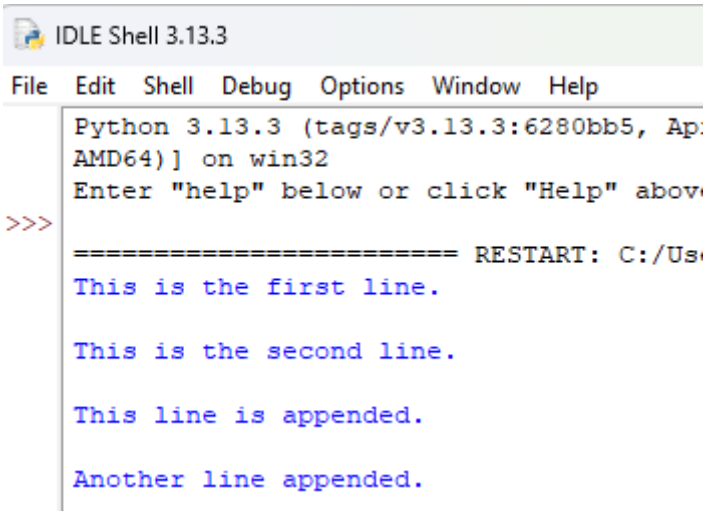


Fig: 3.2.9 new content of a file my_file.txt

Opening the same file again in write mode will overwrite the previous content

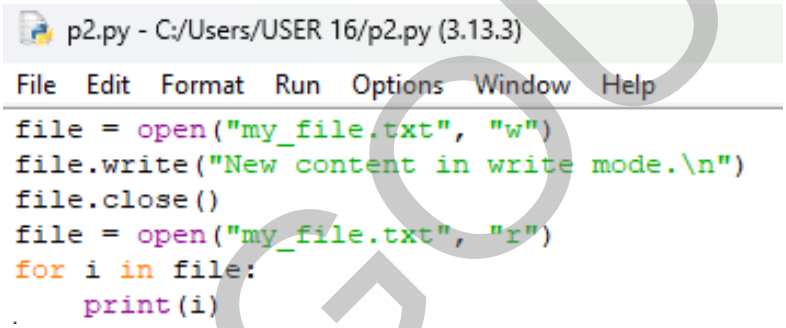


Fig: 3.2.10 File open in write mode, to see the content open the file in read mode.

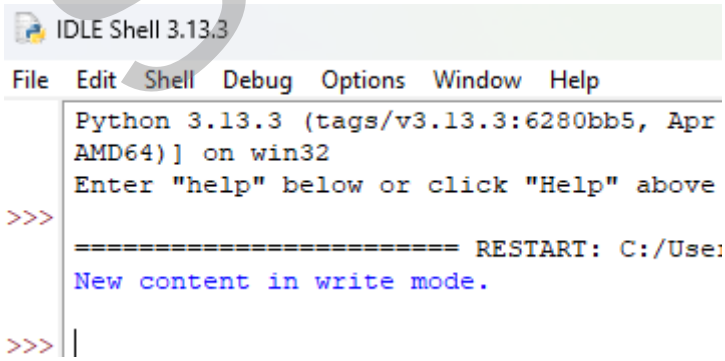


Fig: 3.2.11 new content of a file my_file.txt, previous appended data is erased from the file

Writelines()

It is an efficient way to write multiple lines to a file.

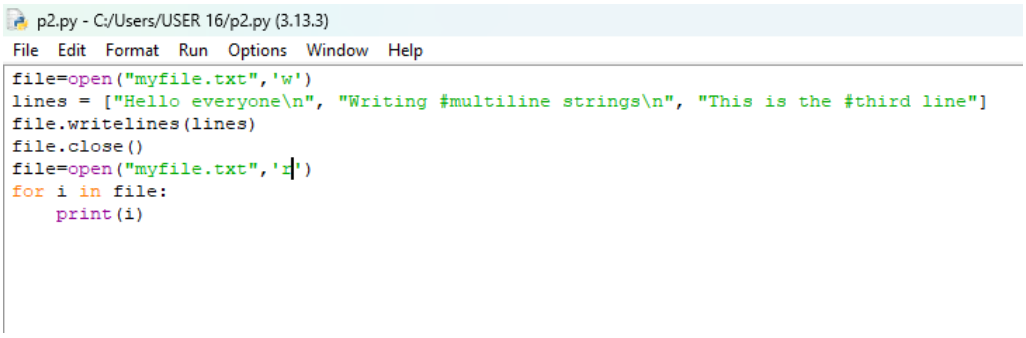


Fig: 3.2.12 An example of writelines()

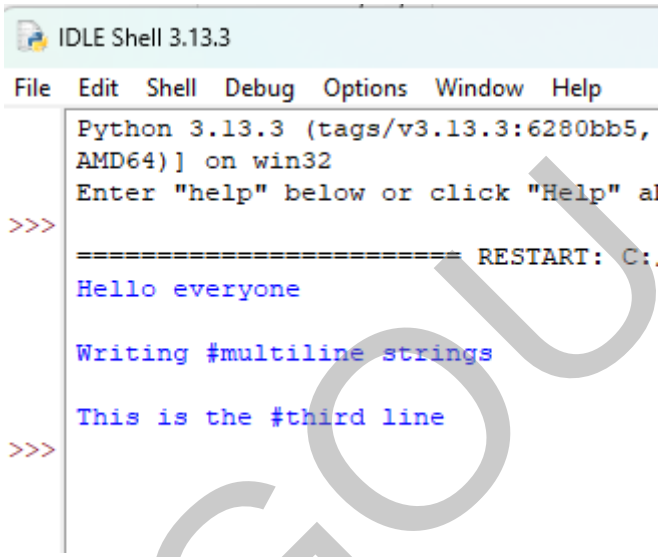


Fig: 3.2.13 content of a file my_file.txt

3.2.5 Reading data from a file using read () mode

In the program above, the content of a file is read using a `for` loop. Another way to read the file content is by using the `read()` method. In Python, there are various methods for reading files. Use file.read() to extract a string containing every character in the file.

6. read() - Read the entire content of a file.

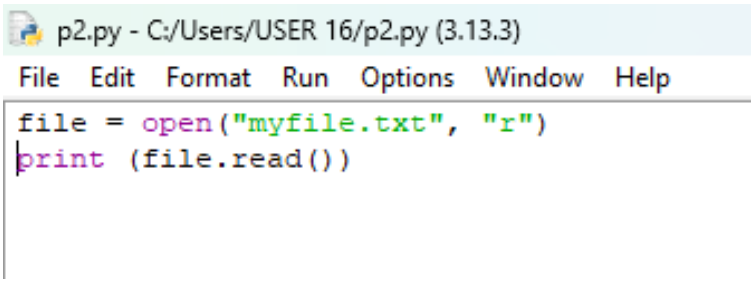


Fig: 3.2.14 File open in read mode and content display using read()

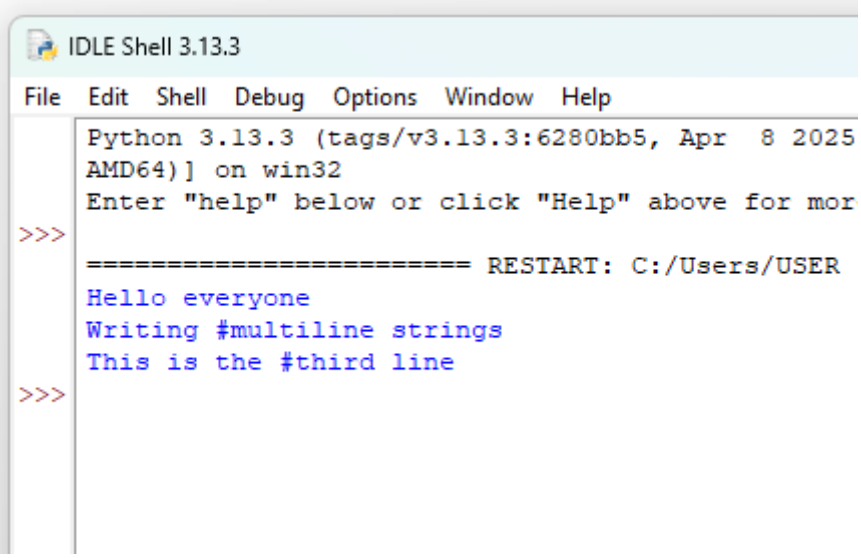


Fig: 3.2.15 Content of myfile.txt

7. read(n)- Read the specified number of characters.

The following code will tell the interpreter to read the first five characters of any stored data and return them as a string, which is another way to read a file:

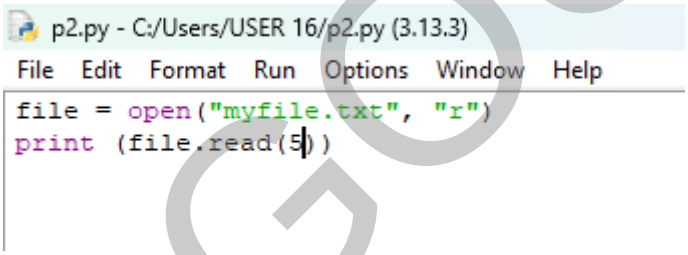


Fig: 3.2.16 File open in read mode and extract 5 characters using read(5)

Fig: 3.2.17 Content of myfile.txt

8. readline(n)

This method reads one complete line from a file where each line terminates with a newline (\n) character. It can also be used to read a specified number (n) of bytes of data from a file but maximum up to the newline character (\n).

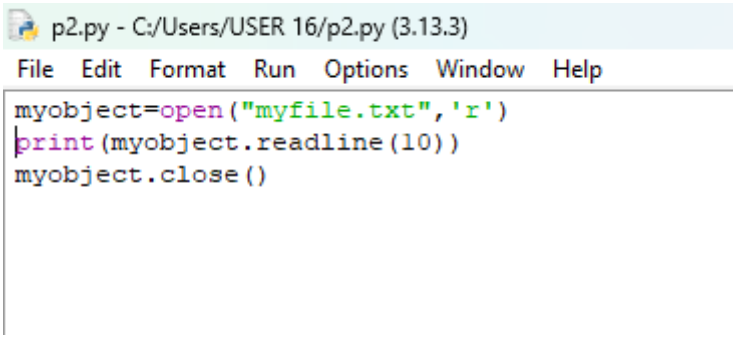


Fig: 3.2.18 File open in read mode and extract 10 characters using readline(10)

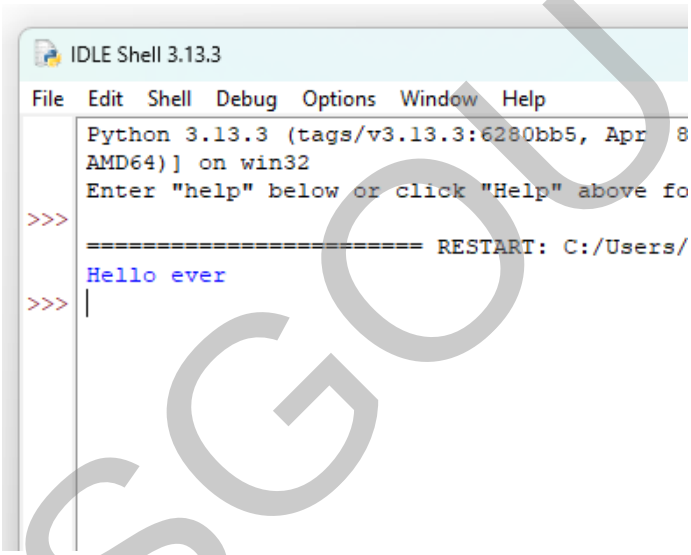


Fig: 3.2.19 Content of myfile.txt

9. readline()

If no argument or a negative number is specified, it reads a complete line and returns string.

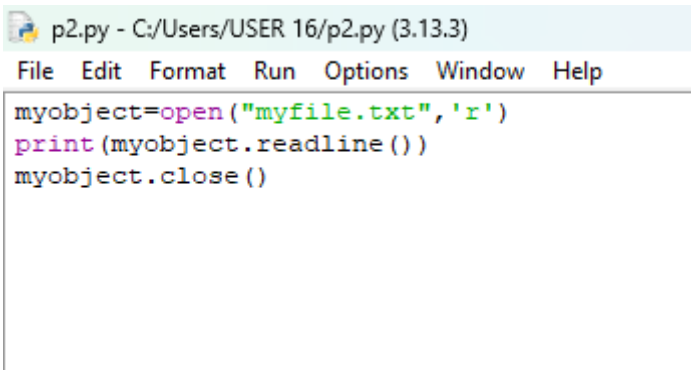


Fig: 3.2.20 File open in read mode and display the content using readline()

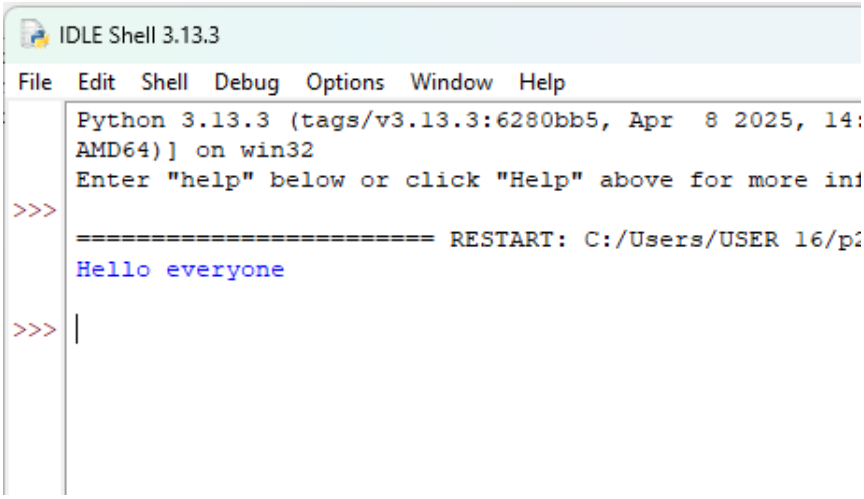


Fig: 3.2.21 Content of myfile.txt

10. readlines() - reads all the lines

To read all the lines in a file, use `readlines()`. The code is listed below:

file = open("myfile.txt", 'r')

f=file.readlines()

However, `readlines()` includes the newline character (`\n`) at the end of each line. To remove these newline characters from the output, apply `rstrip('\n')` to each line in the list returned by `readlines()`."

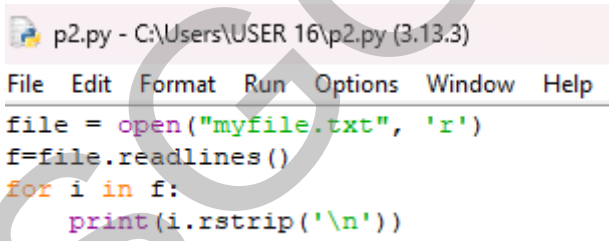


Fig: 3.2.22 File open in read mode, display content using readlines() and rstrip()

To remove the newline character, apply `rstrip('\n')` to each string element. Otherwise, `\n` will remain at the end of each line.

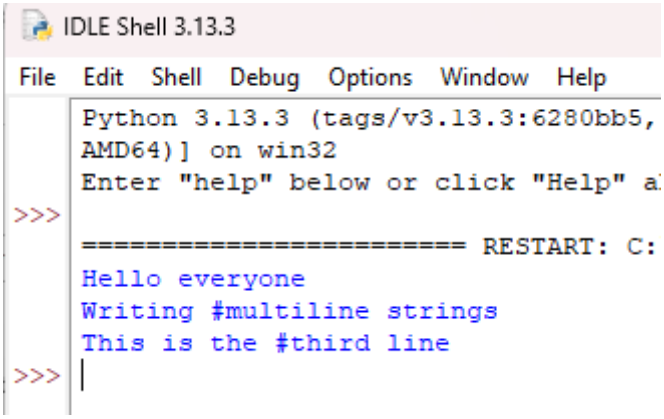


Fig: 3.2.23 Content of myfile.txt

3.2.6 Cursor Positioning Methods

The functions that we have learnt till now are used to access the data sequentially from a file. Python offers the `seek()` and `tell()` functions to enable random data access within a file.

The seek() method

The `seek()` method provides the functionality to move the file pointer to a specific location within a file.

Syntax is: `file_object.seek(offset[, reference_point]).`

Here, `offset` specifies the number of bytes to move the file pointer. The optional `reference_point` argument determines the starting position from which this offset is calculated. It can take one of three values: `0` to indicate the beginning of the file, `1` for the current position of the file pointer, and `2` for the end of the file. If `reference_point` is not provided, it defaults to `0`, meaning the `offset` is counted from the beginning of the file.

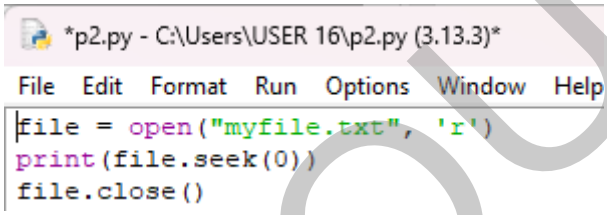


Fig: 3.2.24 File seek()

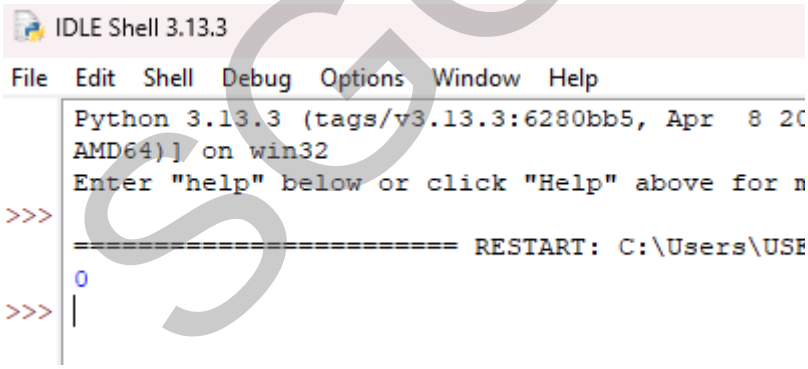


Fig: 3.2.25 Output of file seek()

The cursor will then be moved to index position 0, and file reading will once more begin at the beginning.

3.2.6.1 The tell() method

The `tell()` method returns the current position of the file pointer within the file as an integer. This integer represents the number of bytes from the very beginning of the file up to the current location. The syntax for using `tell()` is:

```
file_obj.tell()
```

The tell() method in Python prints the current position of our cursor.

```
p2.py - C:\Users\USER 16\p2.py (3.13.3)
File  Edit  Format  Run  Options  Window  Help
file = open("myfile.txt", 'r')
print(file.seek(10))
print(file.tell())
file.close()
```

Fig: 3.2.26 File seek() and tell()

```
IDLE Shell 3.13.3
File  Edit  Shell  Debug  Options  Window  Help
Python 3.13.3 (tags/v3.13.3:6280bb5, A
AMD64)] on win32
Enter "help" below or click "Help" abo
>>>
========== RESTART: C:\U
10
10
>>>
```

Fig: 3.2.27 Output of file seek() and file tell()

```
p2.py - C:\Users\USER 16\p2.py (3.13.3)
File  Edit  Format  Run  Options  Window  Help
print("Learning to move the file object")
f=open("myfile.txt","r+")
str=f.read()
print(str)
print("Initially, the position of the file object is: ",f. tell())
f.seek(0)
print("Now the fi le object is at the beginning of the file: ",f.t
f.seek(10)
print("We are moving to 10th byte position from the beginning of f
print("The position of the file object is at", f.tell())
str=f.read()
```

Fig: 3.2.28 Example of seek() and tell()

```
IDLE Shell 3.13.3
File  Edit  Shell  Debug  Options  Window  Help
Python 3.13.3 (tags/v3.13.3:6280bb5, Apr  8 2025, 14:47:33) [MSC v
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\USER 16\p2.py ==========
Learning to move the file object
Hello everyone
Writing #multiline strings
This is the #third line
Initially, the position of the file object is:  67
Now the fi le object is at the beginning of the file:  0
We are moving to 10th byte position from the beginning of file
The position of the file object is at 10
yone
Writing #multiline strings
This is the #third line
>>>
```

Fig: 3.2.29 Output of seek() and tell()

3.2.7 File handling functions in python

Table 3.2.2 File handling functions

Function	Description	Syntax
open()	Open a file	open(filename, mode)
close()	Close the file	file_object.close()
read()	Reads the entire file content (or up to size bytes/characters)	file_object.read(size)
readline()	Reads one line from the file.	file_object.readline()
readlines()	Reads all lines from the file into a list.	file_object.readlines()
write()	Writes a string (or bytes) to the file.	file_object.write(string)
writelines()	Writes a list of strings (or bytes) to the file.	file_object.writelines(list_of_strings)
tell()	Returns the current position of the file pointer (in bytes)	file_object.tell()
seek()	Changes the position of the file pointer to a specific offset.	file_object.seek(offset[, reference point])

Recap

- Python provides file handling and enables users to read and write files as well as perform a variety of other operations on files.
- Python enables programs to interact with files for storing and retrieving data.
- The `open()` function is used to establish a connection with a file.
- The `with` statement provide another way to open a file.
- The `mode` argument in `open()` specifies the intended operation (read, write, append).
- Read mode (`'r'`) allows only reading from a file.
- Write mode (`'w'`) allows writing and overwrites existing file content.
- Append mode (`'a'`) adds new data to the end of a file.

- The close() method is essential to release system resources after file operations.
- The write() method writes a single string to a file.
- The writelines() method efficiently writes a sequence of strings to a file.
- The read() method reads the entire content of a file as a single string.
- The read(n) method reads a specified number of characters from a file.
- The readline() method reads one line at a time from a file, including the newline character.
- The readlines() method reads all lines from a file and returns them as a list of strings.
- The rstrip('\n') method is used to remove trailing newline characters from strings read from a file.
- The seek() method allows changing the position of the file pointer for random access.
- The tell() method returns the current position of the file pointer within the file.

Objective Type Questions

1. What is the primary purpose of file handling?
2. Which function opens a file in Python?
3. What mode is for reading only?
4. Which mode overwrites file content?
5. What mode adds data to the end of a fle?
6. Which method closes a file?
7. Which method writes a single string?
8. Which method writes multiple lines?
9. Which method reads the entire file?
10. Which method reads one line?
11. Which method reads all lines into a list?

12. Which method removes trailing characters?

13. Which method moves the file pointer?

14. Which method tells the current pointer position?

15. After opening a file object named `my_file`, what command would you use to read and print the first line?

16. To open a file named "report.txt" for reading, what Python command would you use?

Answers to Objective Type Questions

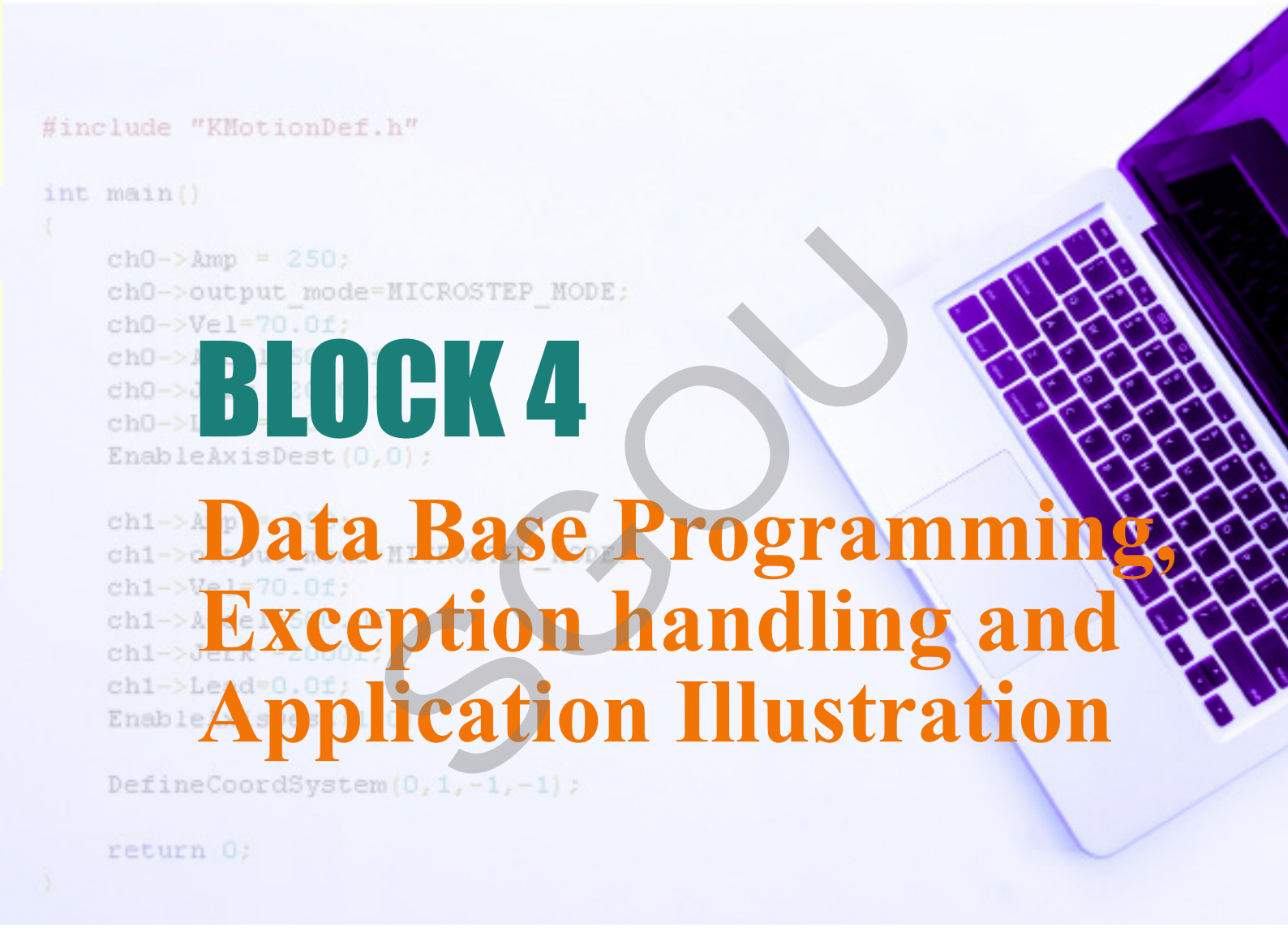
1. **storage**
2. **open()**
3. 'r'
4. 'w'
5. 'a'
6. close()
7. write()
8. writelines()
9. read()
10. readline()
11. readlines()
12. rstrip()
13. seek()
14. tell()
15. **print(my_file.readline())**
16. **open("report.txt", "r")**

References

1. Matthes, E., 2016, Python Crash Course, No Starch Press
2. Lutz, M., 2013, Learning Python, O\'Reilly Media' '
3. Beazley, D., 2013, Python Cookbook, O\'Reilly Media'
4. https://docs.python.org/3/tutorial/
5. https://wiki.python.org/moin/
6. https://realpython.com/

Suggested Reading

1. Ray, B., (2020), Python Programming: A Modern Approach, Packt Publishing
2. Matthes, E. (2019). Python Crash Course: A Hands-On, Project-Based Introduction to Programming (2nd ed.). No Starch Press.
3. Lutz, M., (2019), Programming Python, O\'Reilly Media'
4. Brett Slatkin, 2015, Effective Python, Addison-Wesley Professional



BLOCK 4

Data Base Programming, Exception handling and Application Illustration

UNIT 1 Database programming and Exception handling

Learning Outcomes

At the end of this unit, the learner will be able to:

- understand the role and importance of databases in Python programming.
- become familiar with the concepts of tables and records within a database.
- explore the process of interacting with databases using Create, Retrieve, Update, and Delete (CRUD) operations.
- learn to manage and handle various types of exceptions in Python effectively.

Prerequisites

Python, as a high-level programming language, provides strong support for working with a variety of databases. It allows developers to connect to databases and execute queries programmatically, eliminating the need to manually input raw commands through a terminal or the database's native shell. The only prerequisite is that the appropriate database system must be installed on the computer.

Consider a scenario where you're watching a video on YouTube, and suddenly the internet connection is lost. As a result, the video stops playing. This kind of interruption is an example of an exception. Effectively handling such exceptions is crucial for ensuring smooth program execution and user experience.

Exceptions are unforeseen events or errors that occur while a program is running. They may result from various causes, such as incorrect input, missing resources, or bugs in the code. Python offers a powerful exception handling framework that enables developers to detect and manage these errors gracefully. This helps prevent abrupt program termination and allows for more resilient and user-friendly applications.

Key words

Database, SQL, Create, Retrieve, Update, and Delete (CRUD) statements , finally, exception, try

Discussion

Introduction

This unit begins with database programming basics. We have written and executed many programs in the previous classes. For example, calculator application. While running the program, we have given the numbers to add. After that, the programs will show the output. When we run the program again, we need to input the data(numbers to add) again. What happened to the data we entered earlier? The data was saved temporarily during the execution of the program. We have to use a database or file to save the data for future use. Database software will save the data permanently. Python supports file handling. The file is the place to store data or information. We can create, update, delete, search and do other file handling operations in Python. Using a file handling method in any programming language has many limitations and issues. The alternate way to save the data is using database programs. The popular database programs are MySQL, Oracle, Microsoft Database software that is used to save and manipulate the data. A database management system (DBMS) is a comprehensive database application to work with the database. DBMS allows the users to store, retrieve, update and manage the data in an organized and optimized manner. The language used to store, retrieve, update and manage the data is called Structured Query Language (SQL). It is used by databases such as Microsoft Access, among others.

MySQL is an open-source relational database management system. In this course, we will discuss MySQL. Some of the applications that use MySQL are Twitter, LinkedIn, Facebook, YouTube, etc. For example, in the Facebook sign up process, we will fill the following form and click on signup.

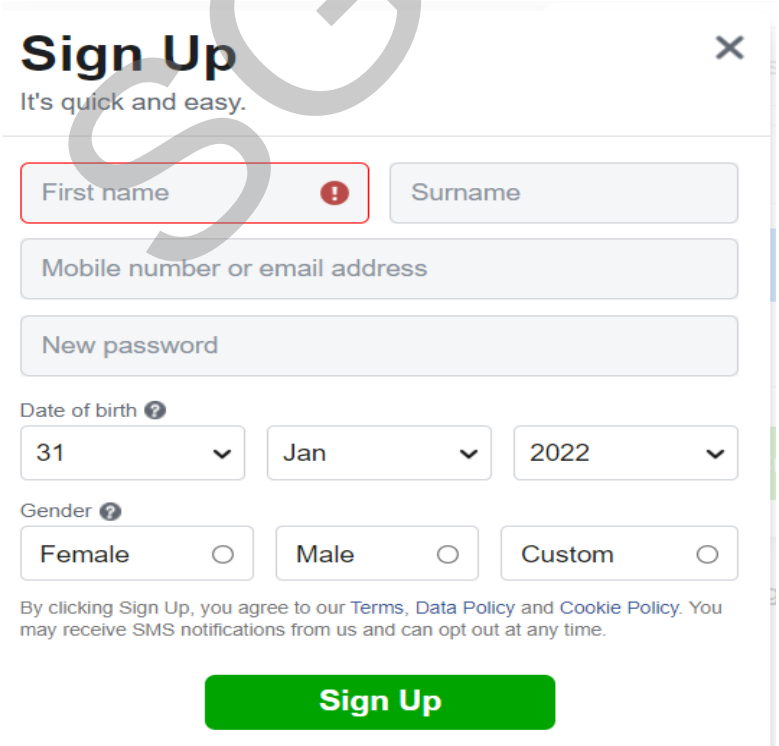


Fig. 4.1.1 Sign in page

The application will verify the data entered and save the data to the database. The verification part is done by the scripting or programming language and the saving data to the database is done by the SQL. In conclusion, to develop an application, we need a database program to save the data, a programming language, or a scripting language and other components (will learn in different courses).

Upon clicking on Sign Up, the Facebook application will send the user data to Facebook's server on which the database resides.

A database is a collection of data organized as tables, records, and fields. For example, the University database has a student table, Course table, Examination table, etc. The student table consists of student records. The following table has four fields and 2 records.

Table 4.1.1 Student Table

Student ID	Student name	City	Phone number
SNOU123	KKG	Kollam	97642789
SNOU222	Shan	Trivandrum	89902233

Install MySQL on your computer.

4.1.1 Example of database programming in Python

Program to create a database. Type the following code in Python IDLE, save and run.

```
import mysql.connector

MyFirstDB_Connection = mysql.connector.connect(host="localhost",user="root", password="Kxxxxxx!")

mycursor = MyFirstDB_Connection.cursor()

mycursor.execute("CREATE DATABASE MyFirstDB")
```

The first step is to import the mysql.connector. This connector is a self-contained Python driver for communicating Python with MySQL servers. Second step is to connect Python with MySQL.

MyFirstDB_Connection is the name of the connection. It could be any name. The host is the place the server is located. In our case, MySQL is installed on the same computer we are working with. Hence the host is the localhost or IP address of the local host. During the Facebook sign-up process the connection will be made to Facebook's server on which their database is created and the host will be the name of the Facebook's domain or IP address. By default, the username is root. We can create users in MySQL and use it. Password is the password set for the user root. MySQL cursor interacts with the MySQL server to execute operations such as SQL statements. The name of the database created in the above example is MyFirstDB and the name of the cursor is my

cursor. It could be any name. The cursor permits row-by-row processing of the result sets. It is used to fetch the results returned from a query.

The following program will display all databases created.

```
import mysql.connector
MyFirstDB_Connection = mysql.connector.connect(host= "localhost",user =
"root", password= "Kxxxxxl")
myc = MyFirstDB_Connection.cursor( )
myc.execute("show databases")
for k in myc:
        print(k)
```

This program will connect with MySQL, execute the SQL command to show databases, and display the database names retrieved and stored in the cursor one by one using a for loop. k is a variable name. It could be any name.

The following is a program to create a table in the database. Remember, the name of the database we created is MyFirstDB.

```
import mysql.connector
MyFirstDB_Connection = mysql.connector.connect(host = "localhost",user = "root",
password = "Kxxxxxl", database = " MyFirstDB")
myc = MyFirstDB_Connection.cursor()
myc.execute("CREATE TABLE Student(StudentID VARCHAR(100),
Student_NameVARCHAR(255),\CityVARCHAR(100),Phone_NoVARCHAR(10))")
```

Note: \ is used to write the SQL statement in multiline. Varchar(100) represents a variable character data type that can store a maximum of 100 characters. (The name John contains 4 characters)

There are different types of data in MySQL. For example, INTEGER will be used to store the number of students

Refer MySQL manual https://dev.mysql.com/doc/refman/8.0/en/data-types.html

The following program displays the tables created in the MyFirstDB database.

```
import mysql.connector
# Establishing connection to the MySQL server and specific database
MyFirstDB_Connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Kxxxxxl",  # Replace with your actual password
    database="MyFirstDB"  # Ensure this database exists
)
# Creating a cursor object
myc = MyFirstDB_Connection.cursor( )
# Executing a SQL query to show all tables in the selected database
myc.execute("SHOW TABLES")
# Printing the names of the tables
for t in myc:
    print(t)
```

The following program inserts data into the tables.

```
import mysql.connector
# Establishing connection to the database
MyFirstDB_Connection = mysql.connector.connect(
    host="localhost",
    user="root",
   password="Kxxxxxl",  # Replace with your actual
password
    database="MyFirstDB"
)
# SQL insert statement
Insert_Student = "INSERT INTO student (StudentID,
```

```
Student_Name, City, Phone_NO) VALUES (%s, %s, %s, %s)"
# Data to be inserted
Student_Data = ("sgou1", "Diya", "Kollam", "9283458929")
# Creating a cursor object
myc = MyFirstDB_Connection.cursor()
# Executing the insert query
myc.execute(Insert_Student, Student_Data)
# Committing the transaction to save changes
MyFirstDB_Connection.commit( )
# Closing the connection
myc.close( )
MyFirstDB_Connection.close( )
```

The following program retrieves the data from the table and displays it.

```
import mysql.connector
# Establishing connection to the database
MyFirstDB_Connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Kxxxxxl",  # Replace with your actual password
    database="MyFirstDB"
)
# Creating a cursor object
myc = MyFirstDB_Connection.cursor( )
# Executing the SQL query to fetch all records from the Student table
```

```
myc.execute("SELECT * FROM Student")
# Fetching all results
Result = myc.fetchall( )
# Iterating over the result and printing each row
for k in Result:
    print(k)
# Closing the connection
myc.close()
MyFirstDB_Connection.close( )
```

The following program reads the student data from the input device and inserts it into the table.

```
import mysql.connector
# Getting user input
SID = input("Enter student ID: ")
Sname = input("Enter student name: ")
city = input("Enter city name: ")
phone = input("Enter phone number: ")
# Establishing connection to the database
MyFirstDB_Connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Kxxxxxl",  # Replace with your actual password
    database="MyFirstDB"
)
# SQL Insert query with placeholders
Insert_Student = "INSERT INTO student (StudentID, Student_Name, City, Phone_
NO) VALUES (%s, %s, %s, %s)"
```

```
# Tuple of values to insert
Student_Data = (SID, Sname, city, phone)
# Creating a cursor object
myc = MyFirstDB_Connection.cursor( )
# Executing the insert query
myc.execute(Insert_Student, Student_Data)
# Committing the transaction
MyFirstDB_Connection.commit( )
# Confirmation message
print("Student record inserted successfully.")
# Closing cursor and connection
myc.close( )
MyFirstDB_Connection.close( )
```

In this program, four variable names are used to input the data.

Student_Data = (SID,Sname,city,phone)

Note that the variable names in the list are not in quotes(‘ ’)

We click on the sign-up button after entering the data in the Facebook registration process. The data we entered in the registration form will be saved to the table.

Activity: Create a login table and insert data into that.

Username	**Password**
Krishna	56Urte
Joy	J9otrw2!

Note: username must be the primary key. The values will not be repeated.

CREATE TABLE login(username VARCHAR(100) PRIMARY KEY, password VARCHAR(255))

The following is a sample code for the Login module.

```
import mysql.connector

User_Name = input("Enter user name : ")

pass = input("Enter password : ")

MyFirstDB_Connection=mysql.connector.connect(host= "localhost",user= "root",
password= "Kxxxxxl",database = "MyFirstDB")

myc.execute("SELECT * FROM login where username=%s and
password=%s", (User_Name,pass))

Result = myc.fetchone() if Result:

print("Login successful")

else:

print("Invalid user name or password")
```

The user will be prompted to enter the username and password. Enter the username and password to the SQL query as shown in the code. **User_Name and Pass** are two variable names and **username** and **password** are the fields in the **login table**.

Activity: Insert the username Krishna two times and observe the result.

We can use the DELETE query to delete a record and UPDATE to make changes to the existing records.

DELETE FROM student WHERE city = 'Kollam'

UPDATE student SET city = ' Kottayam' WHERE Student_ID = 'SNOU123'

Note: Refer to the MySQL Manual to learn more about SQL.

Python supports various databases like MySQL, SQLite, Sybase, Oracle, etc. Python also supports the NoSQL database MongoDB. NoSQL ("not only SQL") is a non-tabular database and stores data differently than relational tables such as MySQL, SQLite, Sybase, Oracle

4.1.2 Introduction to Exception Handling

In Python, there are two different types of errors: syntax errors and exceptions. Errors are issues in a program that cause it to halt during execution. On the other hand, exceptions are raised when internal events take place that alter the program's usual course.

4.1.2.1 Difference between Exceptions and Syntax Errors

Syntax errors and exceptions are both types of errors in Python, but they occur in different situations. A syntax error happens when the code violates the rules of the Python language such as a missing colon, unmatched parentheses, or incorrect indentation and prevents the program from running at all. These errors must be fixed before the code

can execute. On the other hand, an exception is an error that occurs during the execution of a syntactically correct program. Examples include dividing by zero, accessing a file that doesn't exist, or using a variable before assigning a value.

1. Syntax Error

As the name implies, this error results from incorrect syntax in the code. It results in the program's termination.

initialize the amount variable amount = 20000

if(amount > 2999) print("You are eligible")

Output:

SyntaxError: invalid syntax

2. Exceptions

When the program is syntactically sound but the code produces an error, exceptions are raised. Although the application continues to run while experiencing this error, the typical course of the program is altered.

marks = 10000

a = marks / 0

print(a)

Output:

ZeroDivisionError: division by zero

The attempt to divide a number by zero in the example above triggered the ZeroDivisionError.

4.1.3 Try and Except Statement – Catching Exceptions

In Python, exceptions are caught and dealt with using the try and except commands. The try and except clauses are used to contain statements that can raise exceptions and statements that handle such exceptions.

a = [1, 2, 3]

try:

print ("Second element = %d" %(a[1])) print ("Fourth element = %d" %(a[3]))

except:

print ("An error occurred")

Output

Second element = 2

An error occurred

The statements that could result in the error are contained inside the try statement in the example above (second print statement in our case). a[3] is out of range (since the list has only 3 elements with indices 0, 1, 2), which raises an IndexError. The except statement then handles this exception.

4.1.3.1 Catching Specific Exception

To provide handlers for various exceptions, a try statement may contain more than one except clause. We may, for instance, add IndexError to the code shown above. The standard syntax for adding certain exceptions –

```
def fun(a):
    if a < 4:
        # Throws ZeroDivisionError for a = 3
        b = a / (a - 3)
        print("Value of b =", b)
    if a >= 4:
        # Throws NameError because 'b' is not defined in this block
        print("Value of b =", b)
try:
    fun(3)
    fun(5)
# Handling multiple exceptions separately
except ZeroDivisionError:
    print("ZeroDivisionError Occurred and Handled")
except NameError:
    print("NameError Occurred and Handled")
```

Output

ZeroDivisionError Occurred and Handled

If you comment on the line fun(3), the output will be NameError Occurred and Handled

The output above is so because as soon as python tries to access the value of b, NameError occurs.

4.1.4 Finally Keyword in Python

The finally keyword is available in Python, and it is always used after the try and except blocks. The final block is always executed after the try block has terminated normally

or after the try block has terminated for some other reason.

Syntax:

```
try:
# Some Code.... except:
# optional block
# Handling of exception (if required) else:
# execute if no exception finally:
# Some code    (always executed)
```

Example:

```
try:
k = 5/0
# raises divide by zero exception.
print(k)
# handles zero division exception
 except  ZeroDivisionError:
print("Can't divide by zero")
finally:
# this block is always executed
# regardless of exception generation.
print('This is always executed')
```

Output:

```
Can't divide by zero
This is always executed
```

Recap

Database Programming Basics

- Data entered during program execution is temporary.
- Files or databases are needed for permanent data storage.
- Python supports file handling (create, update, delete, search).
- File handling has limitations → Databases are better alternatives.
- DBMS (Database Management System) helps store, retrieve, update, manage data.
- SQL (Structured Query Language) is used to interact with DBMS.
- MySQL is an open-source RDBMS.
- Applications like Facebook, Twitter, LinkedIn, YouTube use MySQL.
- Python uses mysql.connector to connect with MySQL.

Database Concepts

- Database = collection of data organized as tables, records, fields.
- Example: University database → Student table, Course table, etc.
- Tables consist of fields (columns) and records (rows).

Connecting Python with MySQL

- Use mysql.connector.connect() with parameters: host, user, password, database.
- cursor() object is used to execute SQL queries.
- for loop used to fetch and print data from cursor.

Basic SQL Commands in Python

- **Create database**: Use Python to create MySQL database.
- Show databases: myc.execute("SHOW DATABASES")
- Create table: Use SQL CREATE TABLE with proper data types (VARCHAR, INTEGER, etc.).
- Show tables: myc.execute("SHOW TABLES")
- Insert data: INSERT INTO table VALUES (...) with placeholder %s.

- Select data: SELECT * FROM table
- Input from user: Use input() to collect data, then insert into table.
- **Primary key**: Ensures uniqueness in table fields (e.g., username).

Introduction to Exception Handling

- **Error**: Issue halts program execution.
- **Exception**: Runtime anomaly alters normal flow.

Syntax Error vs Exception

- **Syntax Error**: Violates Python syntax (e.g., missing colon, wrong indentation).
- **Exception**: Error during execution of syntactically correct code (e.g., division by zero).

Try and Except

- try block: Code that might raise an exception.
- except block: Handles the exception.

Catching Specific Exceptions

- Use multiple except blocks for specific exceptions.
- Example: ZeroDivisionError, NameError, IndexError

Finally Block

- Always runs whether or not an exception occurs.

Objective Type Questions

1. What is the default username in MySQL?
2. Which keyword is used to handle exceptions in Python?
3. What data type is used to store variable-length strings in MySQL?
4. Which Python module is used to connect with MySQL?
5. What kind of database is MongoDB?
6. Which block in Python is always executed in exception handling?

7. Which MySQL command lists all databases?
8. What type of error occurs if you miss a colon in Python syntax?
9. Which query is used to change existing records in a table?
10. What is the primary key constraint used for in a table?

Answers to Objective Type Questions

1. root
2. except
3. VARCHAR
4. mysql.connector
5. NoSQL
6. finally
7. SHOW
8. SyntaxError
9. UPDATE
10. Uniqueness

Assignments

1. How to install database in python
2. Create Employee database with following field employee id, employee name, designation, address, date of birth, basic pay
3. Explain different types of error in python.
4. Difference between exception and syntax error

Suggested Reading

1. Matthes, E. (2019). Python Crash Course: A Hands-On, Project-Based Introduction to Programming (2nd ed.). No Starch Press.

2. Ramalho, L. (2015). Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media.

3. Bader, D. (2017). Python Tricks: A Buffet of Awesome Python Features. Dan Bader Press.

UNIT 2 Application Illustration

Learning Outcomes

The students will be able:

- recall the steps to create a basic calculator application using the Tkinter library.
- list the components needed to develop a console-based chat application.
- identify the functions used in building a currency converter and random password generator.
- define the logic behind simple applications like BMI calculator and number guessing game.
- state the basic structure and purpose of utility apps such as a word counter, reminder app, and electricity/water bill calculator.

Prerequisites

Skill enhancement is essential in every learner's journey, especially in today's technology-driven world. Gaining basic knowledge in computer programming is important, as it allows students to understand how everyday applications are created. In particular, learning Python helps build a strong foundation for creating simple yet useful tools like calculators, converters, and games. Understanding basic programming concepts such as variables, loops, conditions, and functions is necessary before starting to build these applications.

Familiarity with programming also helps students explore how software is developed in different fields. For this unit, knowing how to write simple Python programs, use the Tkinter library for creating windows and buttons, and build logic for small applications will make learning easier and more effective. With these skills, students can confidently start creating their own applications such as a BMI calculator, reminder app, or number guessing game.

Keywords

Tkinter, GUI, Buttons, Grid, Menu, BMI Calculator

Discussion

4.2.1 Introduction to Python Applications

Python offers a wide array of applications, making it one of the most versatile and beginner-friendly programming languages. Its simple syntax and powerful libraries allow users to develop everything from basic utility tools to complex software systems. Learning to code in Python by starting with small, manageable projects is an effective approach to building both confidence and competence. These hands-on experiences not only strengthen programming skills but also help learners understand how code translates into real-world functionality.

By creating our own Python applications, we can address everyday problems and personalize tools to suit our specific needs and interests. Whether it's a calculator, a password generator, or a reminder app, developing such projects deepens our practical understanding of the language. This process not only reinforces key concepts like variables, loops, and functions but also introduces us to libraries and techniques commonly used in software development. Ultimately, building personal projects is a valuable step toward Python mastery and encourages creativity, problem-solving, and independent thinking.

4.2.2 Build a Simple Calculator Application that can Perform Basic Arithmetic Operations using Tkinter

Building a simple calculator application using Tkinter is an excellent way to begin learning graphical user interface (GUI) development in Python. This application helps learners understand how to design interactive windows, buttons, and input fields, while also reinforcing basic arithmetic operations such as addition, subtraction, multiplication, and division. By using Tkinter, Python's standard GUI toolkit, students gain hands-on experience in event-driven programming and layout management. This application serves as a foundational step toward creating more advanced desktop applications with user-friendly interfaces.

Sample Source Code

```
import tkinter as tk

def button_click(number):

    current = entry.get()

    entry.delete(0, tk.END)

    entry.insert(0, str(current) + str(number))

def button_clear():
```

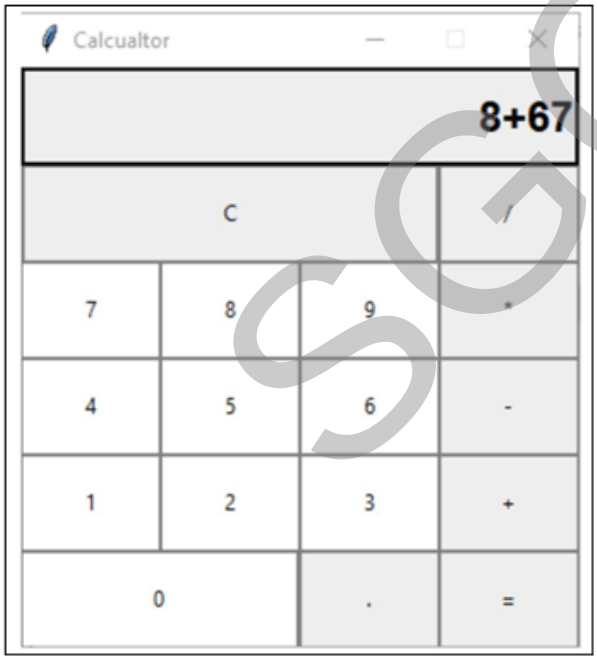
```
    entry.delete(0, tk.END)
def button_add():
    first_number = entry.get()
    global f_num
    global math
    math = "addition"
    f_num = float(first_number)
    entry.delete(0, tk.END)
def button_subtract():
    first_number = entry.get()
    global f_num
    global math
    math = "subtraction"
    f_num = float(first_number)
    entry.delete(0, tk.END)
def button_multiply():
    first_number = entry.get()
    global f_num
    global math
    math = "multiplication"
    f_num = float(first_number)
    entry.delete(0, tk.END)
def button_divide():
    first_number = entry.get()
    global f_num
    global math
    math = "division"
    f_num = float(first_number)
```

```
    entry.delete(0, tk.END)
def button_equal():
    second_number = entry.get()
    entry.delete(0, tk.END)
    if math == "addition":
        entry.insert(0, f_num + float(second_number))
    elif math == "subtraction":
        entry.insert(0, f_num - float(second_number))
    elif math == "multiplication":
        entry.insert(0, f_num * float(second_number))
    elif math == "division":
        if float(second_number) != 0:
            entry.insert(0, f_num / float(second_number))
        else:
            entry.insert(0, "Error")
# Create the main window
root = tk.Tk()
root.title("Simple Calculator")
# Entry widget to display numbers
entry = tk.Entry(root, width=40, borderwidth=5)
entry.grid(row=0, column=0, columnspan=4, padx=10, pady=10)
# Define buttons
button_1 = tk.Button(root, text="1", padx=40, pady=20, command=lambda: button_click(1))
button_2 = tk.Button(root, text="2", padx=40, pady=20, command=lambda: button_click(2))
button_3 = tk.Button(root, text="3", padx=40, pady=20, command=lambda: button_click(3))
button_4 = tk.Button(root, text="4", padx=40, pady=20, command=lambda:
```

```
button_click(4))

button_5 = tk.Button(root, text="5", padx=40, pady=20, command=lambda: button_click(5))

button_6 = tk.Button(root, text="6", padx=40, pady=20, command=lambda: button_click(6))

button_7 = tk.Button(root, text="7", padx=40, pady=20, command=lambda: button_click(7))

button_8 = tk.Button(root, text="8", padx=40, pady=20, command=lambda: button_click(8))

button_9 = tk.Button(root, text="9", padx=40, pady=20, command=lambda: button_click(9))

button_0 = tk.Button(root, text="0", padx=40, pady=20, command=lambda: button_click(0))

button_add = tk.Button(root, text="+", padx=39, pady=20, command=button_add)

button_subtract = tk.Button(root, text="-", padx=41, pady=20, command=button_subtract)

button_multiply = tk.Button(root, text="*", padx=40, pady=20, command=button_multiply)

button_divide = tk.Button(root, text="/", padx=41, pady=20, command=button_divide)

button_equal = tk.Button(root, text="=", padx=91, pady=20, command=button_equal)

button_clear = tk.Button(root, text="Clear", padx=79, pady=20, command=button_clear)

# Put buttons on the screen

button_1.grid(row=3, column=0)

button_2.grid(row=3, column=1)

button_3.grid(row=3, column=2)

button_4.grid(row=2, column=0)

button_5.grid(row=2, column=1)

button_6.grid(row=2, column=2)

button_7.grid(row=1, column=0)
```

```
button_8.grid(row=1, column=1)
button_9.grid(row=1, column=2)
button_0.grid(row=4, column=0)
button_clear.grid(row=4, column=1, columnspan=2)
button_add.grid(row=5, column=0)
button_equal.grid(row=5, column=1, columnspan=2)
button_subtract.grid(row=6, column=0)
button_multiply.grid(row=6, column=1)
button_divide.grid(row=6, column=2)
# Start the GUI
root.mainloop()
```

Output



4.2.3 Develop a simple console-based chat application that allows users to send messages to each other

Developing a simple console-based chat application is a great way to understand the basics of real-time communication between users using Python. This project introduces learners to concepts such as input/output handling, string manipulation, loops, and conditional logic. It also lays the groundwork for understanding client-server interactions,

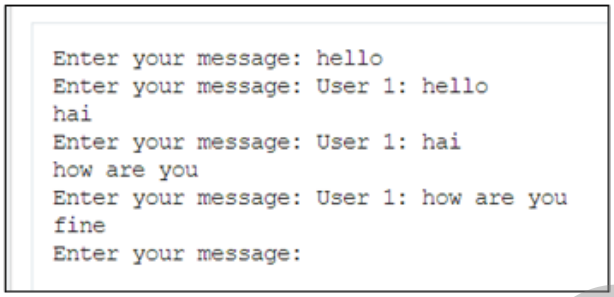
even in a basic form. By building this application, students gain practical experience in creating interactive console programs where users can exchange messages, simulating the core functionality of modern messaging systems in a simplified environment.

Sample Source Code

```
 import threading
class Chat:
    def __init__(self):
        self.messages = []
        self.lock = threading.Lock()
    def send_message(self, sender, message):
        with self.lock:
            self.messages.append((sender, message))
    def get_messages(self):
        with self.lock:
            return self.messages
def send_thread(chat):
    while True:
        message = input("Enter your message: ")
        chat.send_message("User 1", message)
def receive_thread(chat):
    while True:
        messages = chat.get_messages()
        for sender, message in messages:
            print(f"{sender}: {message}")
        chat.messages = []
def main():
    chat = Chat()
    sender_thread = threading.Thread(target=send_thread, args=(chat,))
    sender_thread.start()
```

```
    receiver_thread = threading.Thread(target=receive_thread, args=(chat,))
    receiver_thread.start()
    sender_thread.join()
    receiver_thread.join()
if __name__ == "__main__":
main()
```

Output



4.2.4 To Create Currency Converter

Creating a currency converter program is a practical way to apply basic programming concepts such as user input, conditional statements, and arithmetic operations. This application allows users to convert an amount from one currency to another using predefined exchange rates. By developing this program, learners gain hands-on experience in handling real-world scenarios where accuracy and user interaction are important. It also introduces the idea of data mapping through dictionaries, making it a useful exercise for building foundational programming skills in a meaningful context.

Sample Source Code

```
def display_menu():
    print("1. Convert USD to Euro(EUR)")
    print("2. Convert USD to British Pound Sterling (GBP)")
    print("3. Convert USD to Japenese Yen(JPY)")
    print("4. Convert USD to Canadian Dollar(CAD)")
    print("5. Exit")
# The other four functions that convert currencies accept a value through an argument and return the converted value
def USD_to_EU(value):
```

```
    return value * 0.94
def USD_to_GBP(value):
    return value * 0.79
def USD_to_JPY(value):
    return value * 113
def USD_to_CAD(value):
    return value * 1.33
while True:
    display_menu()
    choice = int(input())
    if choice == 5:
        print("Bye!")
        break
    else:
        amount = float(input("Enter an amount in US dollars: "))
```

Output

```
1. Convert USD to Euro(EUR)
2. Convert USD to British Pound Sterling (GBP)
3. Convert USD to Japenese Yen(JPY)
4. Convert USD to Canadian Dollar(CAD)
5. Exit
4
Enter an amount in US dollars: 34
34.0 USD 45.22 CAD
1. Convert USD to Euro(EUR)
2. Convert USD to British Pound Sterling (GBP)
3. Convert USD to Japenese Yen(JPY)
4. Convert USD to Canadian Dollar(CAD)
5. Exit
```

4.2.5 Random password generator

A Random Password Generator is a useful application that helps users create strong, secure passwords to protect their online accounts and sensitive information. This project introduces learners to key programming concepts such as string manipulation, use of the random module, loops, and user input. By building this application, students learn how to generate passwords with a mix of uppercase and lowercase letters, numbers,

and special characters. It emphasizes the importance of cybersecurity and demonstrates how simple programs can solve real-world problems by enhancing digital safety.

Sample Source Code

```
import random
import string
def generate_password(length=12):
    # Define the character set for the password
    characters = string.ascii_letters + string.digits + string.punctuation
 # Generate the password
    password = ''.join(random.choice(characters) for _ in range(length))
    return password
def main():
    # Get the desired length of the password from the user
    length = int(input("Enter the length of the password: "))
    # Generate the password
    password = generate_password(length)
    # Display the generated password
    print("Generated Password:", password)
if __name__ == "__main__":
    main()
```

Output

```
Enter the length of the password: 10
Generated Password: sN7M=':\jd
```

4.2.6 BMI calculator

A **BMI (Body Mass Index) Calculator** is a simple yet practical application that helps users determine their body mass index based on their height and weight. This project introduces learners to basic arithmetic operations, user input handling, and conditional statements in Python. By developing a BMI calculator, students learn how to apply formulas in code and display meaningful health-related results. It also demonstrates

how programming can be used to create helpful tools that promote personal wellness and everyday decision-making.

Sample Source Code

```
def calculate_bmi(weight_kg, height_m):
    """
    Calculate BMI (Body Mass Index) using weight in kilograms and height in meters.
    Formula: BMI = weight (kg) / (height (m) ** 2)
    """
    return weight_kg / (height_m ** 2)
def main():
    # Get user input for weight and height
    weight_kg = float(input("Enter your weight in kilograms: "))
    height_m = float(input("Enter your height in meters: "))
    # Calculate BMI
    bmi = calculate_bmi(weight_kg, height_m)
    # Display the BMI
    print("Your BMI is:", bmi)
    # Interpret the BMI
    if bmi < 18.5:
        print("You are underweight.")
    elif bmi >= 18.5 and bmi < 25:
        print("Your weight is normal.")
    elif bmi >= 25 and bmi < 30:
        print("You are overweight.")
    else:
        print("You are obese.")
if __name__ == "__main__":
    main()
```

Output

Enter your weight in kilograms: 63 Enter your height in meters: 1.52

Your BMI is: 27.268005540166204

You are overweight.

4.2.7 Number Guessing Game

The **Number Guessing Game** is an engaging and interactive project that introduces learners to fundamental programming concepts in a fun and creative way. In this game, the computer randomly selects a number within a specified range, and the user tries to guess it. With each attempt, the program provides hints to guide the player such as whether the guess is too high or too low until the correct number is guessed. This project helps students practice the use of loops, conditionals, random number generation, and user input handling, making it a great starting point for beginners to build logical thinking and problem-solving skills.

Sample Source Code

```
import random
def number_guessing_game():
    # Generate a random number between 1 and 100
    secret_number = random.randint(1, 100)
    # Number of attempts allowed
    attempts = 0
    max_attempts = 10
    print("Welcome to the Number Guessing Game!")
    print("I'm thinking of a number between 1 and 100. Can you guess it?")
    while attempts < max_attempts:
        try:
            # Get user's guess
            guess = int(input("Enter your guess: "))
            # Check if the guess is correct
            if guess == secret_number:
          print(f"Congratulations! You guessed the number {secret_number} correctly
in {attempts + 1} attempts!")
```

```
            break
        elif guess < secret_number:
            print("Too low! Try again.")
        else:
            print("Too high! Try again.")
        attempts += 1
    except ValueError:
        print("Please enter a valid number.")
if attempts == max_attempts:
    print(f"Sorry, you've run out of attempts. The correct number was {secret_
number}.")
if __name__ == "__main__":
    number_guessing_game()
```

Output

```
Welcome to the Number Guessing Game!
I'm thinking of a number between 1 and 100. Can you guess it
?
Enter your guess: 3
Too low! Try again.
Enter your guess: 7
Too low! Try again.
Enter your guess: 54
Too high! Try again.
Enter your guess: 33
Too high! Try again.
Enter your guess: 25
Too high! Try again.
Enter your guess: 20
Too high! Try again.
Enter your guess: 15
Too low! Try again.
Enter your guess: 18
Too low! Try again.
Enter your guess: 19
Congratulations! You guessed the number 19 correctly in 9 at
tempts!
```

4.2.8 Word counter

A **Word Counter** is a simple yet powerful application that counts the number of words in a given text input. This project introduces learners to essential programming concepts such as string manipulation, text processing, and the use of built-in functions in Python. By developing a word counter, students gain practical experience in working with

user input, splitting strings, and iterating through data. This application is especially useful in educational and writing contexts, helping users keep track of word limits in assignments, articles, or reports.

Sample Source Code

```
def word_counter(text):
    """
    Count the number of words in a given text.
    """
    # Split the text into words based on whitespace
    words = text.split()
    # Return the number of words
    return len(words)
def main():
    # Get input text from the user
    text = input("Enter a sentence or paragraph: ")
    # Count the words in the input text
    word_count = word_counter(text)
    # Display the word count
    print("Word count:", word_count)
if __name__ == "__main__":
    main()
```

Output

```
Enter a sentence or paragraph: good morning all, am i a girl
Word count: 7
```

4.2.9 Reminder app

A **Reminder App** is a practical application designed to help users keep track of important tasks, events, or deadlines by sending timely notifications or alerts. Developing this app allows learners to apply basic programming concepts such as working with dates and times, conditional logic, user input, and loops. It introduces the idea of scheduling and time-based actions, helping students understand how programs can interact with real-

world timelines. This project demonstrates how coding can be used to build useful tools that support daily organization and productivity.

Sample Source Code

```
import datetime
import time
def set_reminder():
    """
    Set a reminder for a specific date and time.
    """
    print("Please enter the reminder details:")
    year = int(input("Year (YYYY): "))
    month = int(input("Month (MM): "))
    day = int(input("Day (DD): "))
    hour = int(input("Hour (HH): "))
    minute = int(input("Minute (MM): "))
    reminder_text = input("Reminder message: ")
    reminder_time = datetime.datetime(year, month, day, hour, minute)
    return reminder_time, reminder_text
def main():
    reminder_time, reminder_text = set_reminder()
    print("Reminder set for:", reminder_time.strftime("%Y-%m-%d %H:%M:%S"))
    while True:
        current_time = datetime.datetime.now()
        if current_time >= reminder_time:
            print("Reminder:", reminder_text)
            break
        time.sleep(10)  # Check every 10 seconds for reminder
if __name__ == "__main__":
    main()
```

Output

Please enter the reminder details

Year (YYYY): 2023

Month (MM): 8

Day (DD): 26

Hour (HH): 9

Minute (MM): 42

Reminder message: Happy Onam

Reminder set for: 2023-08-26 09:42:00

Reminder: Happy Onam

4.2.10 Electricity Water Bill Calculator

An **Electricity and Water Bill Calculator** is a practical application that helps users estimate their monthly utility expenses based on usage. This project introduces learners to fundamental programming concepts such as arithmetic operations, conditional statements, and user input handling. By entering units consumed, the application calculates the total cost according to predefined rate slabs, mimicking real-world billing systems. This not only reinforces logical thinking and decision-making but also demonstrates how programming can be used to solve everyday problems efficiently.

Sample Source Code

```
def calculate_electricity_bill(units):
    """Calculates the electricity bill based on the given units.
    Args:
        units: The number of units consumed.
    Returns:
        The total electricity bill amount.
    """
    if units <= 100:
        bill = units * 10
    elif units <= 200:
        bill = (100 * 10) + (units - 100) * 15
    elif units <= 300:
        bill = (100 * 10) + (100 * 15) + (units - 200) * 20
```

```
    else:
        bill = (100 * 10) + (100 * 15) + (100 * 20) + (units - 300) * 25
    return bill
# Example usage
units_consumed = 250
total_bill = calculate_electricity_bill(units_consumed)
print(f"Total electricity bill for {units_consumed} units: {total_bill}")
units_consumed = 50
total_bill = calculate_electricity_bill(units_consumed)
print(f"Total electricity bill for {units_consumed} units: {total_bill}")
units_consumed = 350
total_bill = calculate_electricity_bill(units_consumed)
print(f"Total electricity bill for {units_consumed} units: {total_bill}")
```

Output

```
Total electricity bill for 250 units: 4000
Total electricity bill for 50 units: 500
Total electricity bill for 350 units: 7750
```

Recap

- A basic calculator using Python's Tkinter library is to perform arithmetic operations and understand GUI development.
- A console-based chat application is defined to practice user input/output and simulate simple message exchange between users.
- A currency converter applications is created by applying arithmetic logic and learning how to work with user-defined exchange rates.
- A random password generator is to understand string manipulation, character sets, and the use of the random module.
- A BMI calculator application using user inputs is to calculate and interpret body mass index, reinforcing the use of formulas and conditional statements.
- A number guessing game, which introduced concepts of randomness, loops, and logical feedback mechanisms.
- A word counter application is to practice text processing and string splitting techniques for counting words in user input.
- A reminder app is defined to learn basic scheduling and apply date/time functions for managing tasks and deadlines.
- An electricity/water bill calculator, apply is designed to the conditional logic and arithmetic operations to simulate real-world billing scenarios.
- Strengthened core programming skills like variables, functions, conditionals, loops, and user interaction through practical, hands-on projects.

Objective Type Questions

1. What is the purpose of the button_clear() function in the provided Tkinter program?
2. What is the purpose of the Chat class in the provided Python program?
3. What is the purpose of the display_menu() function in the provided Python code?
4. What is the purpose of the generate_password() function in the provided Python code?
5. What is the function named used to count the number of words in the given text?

6. What type of loop is commonly used in a number guessing game to allow multiple guesses?

7. In a console-based chat application, which function is primarily used to get input from the user?

8. What is the main function of a BMI calculator application?

9. In a number guessing game, which Python module is typically used to generate a random number?

10. Which Python library is commonly used to create GUI applications like a calculator?

Answers to Objective Type Questions

1. To clear the entry widget, allowing for input of a new set of numbers or operations.

2. To facilitate communication by storing and managing messages exchanged between users.

3. To present the user with a menu of currency conversion options.

4. The purpose of the generate_password() function is to create a random password of a specified length using a combination of letters (both uppercase and lowercase), digits, and punctuation characters.

5. len().

6. while loop.

7. input().

8. Calculate health index using height and weight.

9. random.

10. Tkinter.

Assignments

1. Develop a simulator that allows students to interact with and observe molecular biology processes.

2. Build a simulator that allows students to simulate buying and selling stocks with real-time market data.

3. Create a tool that helps students understand and practice medical diagnosis based on symptoms.

4. Build a tool for analyzing and visualizing weather data for environmental studies, Use a weather API to fetch real-time weather data based on the user's location or a specified city.

References

1. Matthes, E. (2019). *Python crash course: A hands-on, project-based introduction to programming* (2nd ed.). No Starch Press.

2. Zelle, J. M. (2017). *Python programming: An introduction to computer science* (3rd ed.). Franklin, Beedle & Associates Inc.

3. Sweigart, A. (2020). *Automate the boring stuff with Python: Practical programming for total beginners* (2nd ed.). No Starch Press.

4. Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

5. Rodas de Paz, A., and Martinez Lopez, M. (2018). *Tkinter GUI application development cookbook* (2nd ed.). Packt Publishing.

Suggested Reading

1. W3Schools – Python Tutorial- https://www.w3schools.com/python/.

2. Geeks for Geeks – Python Programming Language - https://www.geeksforgeeks.org/python-programming-language/.

3. Real Python- https://realpython.com/.

4. Programiz – Python Programming- https://www.programiz.com/python-programming.

MODEL QUESTION PAPER SETS

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-I

End Semester Examination

SGB24CS202SE: PYTHON FOR ALL

(CBCS - UG)

2024-25 - Admission Onwards

Time: 2 Hours Max Marks: 45

Section A

Answer any 5 questions. Each carries one mark **(5 x 1 = 5)**

1. What is a flowchart?
2. Write about any 2 Dictionary built-in methods.
3. What does the seek() function do in file handling?
4. Which keyword is used to handle exceptions in Python?
5. Which method removes trailing characters?
6. Name the type of inheritance where a class is derived from a single base class.
7. Which Python keyword is used to include a library?
8. Which translator converts high-level code into machine language?

Section B

Answer any 5 questions. Each carries two marks **(5 x 2 = 10)**

9. Distinguish between Primary and Secondary memory.
10. What is Encapsulation?
11. What is the difference between data and information?

12. Write about any four Splitting functions in NumPy.

13. What is a one-dimensional labeled data structure in Pandas?

14. Define '*finally'* keyword in Python. Explain with an example.

15. What is the purpose of the open() and close() functions in file handling?

16. What is the purpose of the init method?

Section C

Answer any 4 questions. Each carries five marks **(4 x 5 = 20)**

17. Write short notes on different tokens used in Python.

18. Write a simple console-based chat application that allows users to send messages to each other.

19. Write a short note on the Pandas, Matplotlib, SciPy, and Scikit-learn libraries in Python.

20. Explain about any 5 Methods used for List manipulation.

21. Describe the key features of Object-Oriented Programming with suitable examples.

22. Explain the different types of Polymorphism.

Section D

Answer any 1 question. Each carries ten mark **(1 x 10 = 10)**

23. Explain the basic components of a computer system and describe the phases of programming.

24. Define an array in NumPy. Describe one dimensional, two dimensional and N-dimensional array in NumPy with proper syntax and examples. How can you create an array using existing data in NumPy?

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-II

End Semester Examination

SGB24CS202SE: PYTHON FOR ALL

(CBCS - UG)

2024-25 - Admission Onwards

Time: 2 Hours Max Marks: 45

Section A

Answer any 5 questions. Each carries one mark **(5 x 1 = 5)**

1. "Hello" is which kind of data type?
2. Write about classification of data types in python.
3. Name the function used to read a single line from a file.
4. What is the function named used to count the number of words in the given text?
5. Which method tells the current pointer position?
6. What is the term for a class that inherits properties from another class?
7. Which data structure stores data as key-value pairs?
8. What kind of loop iterates over a sequence?

Section B

Answer any 5 questions. Each carries two marks **(5 x 2 = 10)**

9. Draw flowchart to find largest among five numbers.
10. What is Polymorphism?
11. What is the stored program concept?

12. Write the importance of the public access specifier in Python with an example.

13. What is NumPy in python?

14. Define the following:

    a. Exception

    b. Syntax errors

15. Define class and object with an example.

16. Why is the seek( ) method used in Python file handling?

Section C

Answer any 4 questions. Each carries five marks **(4 x 5 = 20)**

17. Explain different operators used in Python.

18. Write a sample program for calculating electricity and water bills.

19. Explain the difference between primitive and non-primitive data structures in Python. Give suitable examples.

20. Write about any five Set Built-in Methods in python.

21. What is inheritance in Python? Explain the different types of inheritance in Python with suitable diagrams.

22. Explain the use of the Pandas and Matplotlib libraries in Python.

Section D

Answer any 1 question. Each carries ten mark **(1 x 10 = 10)**

23. Describe top-down and bottom-up approaches in problem solving with suitable examples.

24. Explain in detail about String manipulations methods in python.

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

NO TO DRUGS തിരിച്ചിറങ്ങാൻ പ്രയാസമാണ്

ആരോഗ്യ കുടുംബക്ഷേമ വകുപ്പ്, കേരള സർക്കാർ

PYTHON FOR ALL

Course Code: SGB24CA202SE

ISBN 978-81-987966-7-7

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841