**Summary –Day5**

**Name:Tejaswini Gokanakonda**

## **Roll no**:**DE142**

## **Date:11-11-2024**

### **Introduction to Python**

* **Definition**: Python is a high-level, interpreted, and general-purpose programming language known for its simplicity and readability. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.
* **Use Cases**: Web development, data analysis, artificial intelligence, scripting, automation, etc.

### **Python and its Features**

* **Simple and Easy to Learn**: Python's syntax is clear and easy to understand, making it a great language for beginners.
* **Interpreted Language**: Python executes code line-by-line, simplifying debugging and testing.
* **Cross-Platform Compatibility**: Python works on various platforms like Windows, macOS, and Linux without modification.
* **Rich Standard Library**: Python has an extensive library of pre-built functions and modules, reducing development time.
* **Open Source**: Python is freely available for use, modification, and distribution.
* **Object-Oriented and Functional**: It supports both OOP (Object-Oriented Programming) and functional programming paradigms.

### **History of Python**

* **Created by**: Guido van Rossum
* **First Released**: 1991
* **Purpose**: Designed as a successor to the ABC language, focusing on readability and ease of use.
* **Versioning**:
  + **Python 2**: Introduced in 2000, with major enhancements like list comprehensions and Unicode support.
  + **Python 3**: Released in 2008, it included many improvements but broke backward compatibility.

### **Writing and Running the First Program**

1. **Creating a Script**: Write a Python program in a text editor or an IDE (Integrated Development Environment).
2. Example:

print("Hello, World!")

**Running the Program:**

* Using the terminal/command line: python filename.py
* In IDEs like VSCode or PyCharm: Click on the "Run" button.

### **Keywords & Identifiers**

* **Keywords**: Reserved words in Python with special meanings (e.g., if, else, while, def). They cannot be used as variable names.
* **Identifiers**: User-defined names for variables, functions, classes, etc. Rules:
  + Must start with a letter (A-Z/a-z) or underscore (\_).
  + Can contain alphanumeric characters and underscores.
  + Case-sensitive (e.g., Variable and variable are different).

### **Variables & Operators**

* **Variables**: Containers for storing data values. They are created when a value is assigned
* **Example**:

x = 10

name = "Teju"

* **Operators**:
  + **Arithmetic**: +, -, \*, /, //, %, \*\*
  + **Comparison**: ==, !=, >, <, >=, <=
  + **Logical**: and, or, not
  + **Assignment**: =, +=, -=, etc.
  + **Bitwise**: &, |, ^, ~, <<, >>

### **Data Types**

1. **Numeric**:
   * **int**: Integer values (e.g., 1, -100)
   * **float**: Floating-point numbers (e.g., 3.14, -0.5)
   * **complex**: Complex numbers (e.g., 1 + 2j)
2. **Sequence**:
   * **str** (String): Sequence of characters (e.g., "Hello")
   * **list**: Ordered, mutable collection (e.g., [1, 2, 3])
   * **tuple**: Ordered, immutable collection (e.g., (4, 5, 6))
3. **Boolean**:
   * **bool**: Represents True or False
   * Commonly used in conditional statements and logical operations.

**Control Structure**

### **If Statement**

* **Explanation**: An if statement evaluates a condition. If it is true, the code block under it executes; if false, it skips the block.
  + **Example**:

num = 10

if num > 5:

print("Greater than 5")

*Output*: "Greater than 5"

### **If-Else Statement**

* **Explanation**: The if-else statement provides a secondary block of code that runs if the condition is false.
  + **Example**:

num = 3

if num % 2 == 0:

print("Even")

else:

print("Odd")

*Output*: "Odd"

### **If-Elif-Else Statement**

* **Explanation**: This structure allows checking multiple conditions. The first true condition’s block will execute; if none are true, the else block executes.
  + **Example**:

score = 85

if score >= 90:

print("Grade A")

elif score >= 75:

print("Grade B")

else:

print("Grade C")

*Output*: "Grade B"

### **Control Structures**

* **Explanation**: Control structures direct the flow of program execution. These include conditional statements (if, if-else), loops (for, while), and others.
  + **Types**:
    - **Sequential**: Executes code line-by-line.
    - **Conditional**: Makes decisions based on conditions (if, if-else).
    - **Loops**: Repeats code blocks until a condition is met (for, while).

### **For Loop**

* **Explanation**: A for loop iterates over a sequence (like a list or string) and executes the block of code for each item in the sequence.
  + **Example**:

for i in range(3):

print(i)

*Output*:

0

1

2

### **While Loop**

* **Explanation**: A while loop executes as long as a specified condition is True.
  + **Example**:

count = 0

while count < 3:

print("Count is:", count)

count += 1

*Output*:

Count is: 0

Count is: 1

Count is: 2

### **Nested Loop**

* **Explanation**: A loop inside another loop, where the inner loop executes for every iteration of the outer loop.
  + **Example**:

for i in range(2):

for j in range(3):

print(f"i = {i}, j = {j}")

*Output*:

i = 0, j = 0

i = 0, j = 1

i = 0, j = 2

i = 1, j = 0

i = 1, j = 1

i = 1, j = 2

### **Break, Continue, Pass**

* **Break**: Exits the loop prematurely.
  + **Example**:

for i in range(5):

if i == 3:

break

print(i)

*Output*: 0, 1, 2

* **Continue**: Skips the current iteration and moves to the next.
  + **Example**:

for i in range(5):

if i == 2:

continue

print(i)

*Output*: 0, 1, 3, 4

* **Pass**: A placeholder that does nothing.
  + **Example**:

for i in range(3):

if i == 1:

pass # Does nothing

print(i)

*Output*: 0, 1, 2

### **Input and Output**

* **Explanation**: input() gets user input; print() displays output.
  + **Example**:

name = input("Enter your name: ")

print("Hello,", name)

*Input*: "Teju"  
*Output*: "Hello, Teju"

### **Introduction to Lists**

* **Explanation**: Lists are ordered, mutable collections of elements.
  + **Example**:

my\_list = [1, 2, 3]

my\_list.append(4)

print(my\_list)

*Output*: [1, 2, 3, 4]

### **List Methods and Slicing**

* **Explanation**: Lists support operations like adding/removing elements and slicing for extracting sublists.
  + **Methods**:

my\_list = [1, 2, 3]

my\_list.append(4) # Adds 4 to the list

print(my\_list)

*Output*: [1, 2, 3, 4]

* + **Slicing**:

print(my\_list[1:3]) # Extracts elements from index 1 to 2

*Output*: [2, 3]

### **Introduction to Dictionaries & Dictionary Methods**

* **Explanation**: Dictionaries store data in key-value pairs.
  + **Example**:

my\_dict = {"name": "Teju", "age": 25}

print(my\_dict["name"])

*Output*: "Teju"

* + **Methods**:

my\_dict["age"] = 26

print(my\_dict.get("age"))

*Output*: 26

### **Introduction to Sets & Set Methods**

* **Explanation**: Sets are unordered collections of unique elements.
  + **Example**:

my\_set = {1, 2, 3}

my\_set.add(4)

print(my\_set)

*Output*: {1, 2, 3, 4}

### **Introduction to Map & Map Methods**

* **Explanation**: The map() function applies a function to every item in an iterable.
  + **Example**:

def square(num):

return num \* num

numbers = [1, 2, 3]

squared = map(square, numbers)

print(list(squared))

*Output*: [1, 4, 9]

### **Pandas for Data Processing**

* **Explanation**:
  + Pandas is a powerful library in Python used for data manipulation and analysis. It provides data structures like **Series** and **DataFrame** to store and manipulate tabular data efficiently.
  + Pandas helps with tasks such as cleaning data, transforming it, and aggregating large datasets with ease. It is widely used in data preprocessing, exploration, and analysis tasks.
* **Basic Operations**:

import pandas as pd

# Creating a DataFrame

data = {'Name': ['Teju', 'Rahul', 'Asha'], 'Age': [25, 30, 27]}

df = pd.DataFrame(data)

print(df)

*Output*:

markdown

Name Age

0 Teju 25

1 Rahul 30

2 Asha 27

### **Reading CSV Data using Pandas**

* **Explanation**:
  + Pandas provides the read\_csv() function to load data from CSV (Comma-Separated Values) files into a **DataFrame** for analysis.
  + It can read local files, remote URLs, and supports various options for delimiter, header management, and more.
* **Example**:

import pandas as pd

# Reading data from a CSV file

df = pd.read\_csv('data.csv') # Ensure 'data.csv' exists in your directory

print(df.head()) # Displays the first 5 rows of the DataFrame

*Output* (assuming data.csv has sample content):

ID Name Age

0 1 Teju 25

1 2 Rahul 30

2 3 Asha 27

...

### **Read Data from CSV Files to Pandas DataFrames**

* **Explanation**:
  + The pd.read\_csv() function reads data into a Pandas **DataFrame**, a two-dimensional data structure similar to a table in a database or a spreadsheet.
  + It can handle a wide variety of parameters, including delimiters, columns to read, missing value handling, etc.
* **Example**:

# Example of reading data with custom delimiter

df = pd.read\_csv('data.csv', delimiter=',') # Default delimiter is comma

print(df.columns) # Print column names

# Reading a subset of columns

df\_subset = pd.read\_csv('data.csv', usecols=['Name', 'Age'])

print(df\_subset.head())

*Output* (assuming columns Name and Age exist):

Index(['ID', 'Name', 'Age'], dtype='object')

Name Age

0 Teju 25

1 Rahul 30

...