Module :6 Python Fundamentals

1. Introduction to Python and its Features (simple, high-level, interpreted language).

Ans. **Python** is a **high-level**, **interpreted**, and **general-purpose** programming language. It was created by **Guido van Rossum** and released in **1991**. Python is designed to be easy to read and simple to implement. It is widely used for web development, data analysis, artificial intelligence, automation, and more.

Python emphasizes **code readability**, using **indentation** to define blocks of code instead of curly braces like other languages (e.g., C, Java).

1. History and evolution of Python.

Ans.

| **Year** | **Version / Event** | **Description** |
| --- | --- | --- |
| **1989** | Idea Begins | Guido van Rossum starts working on Python. |
| **1991** | **Python 0.9.0** | First public release. Included basic features like functions, loops, and exception handling. |
| **1994** | **Python 1.0** | Introduced core programming tools like lambda, map, filter, and reduce. |
| **2000** | **Python 2.0** | Introduced list comprehensions and garbage collection (via reference counting). |
| **2008** | **Python 3.0** ("Py3k") | Major redesign; not backward compatible with Python 2. Improved syntax and Unicode support. |
| **2020** | End of Python 2 support | Python 2 reached its end of life on January 1, 2020. |
| **2023+** | Latest versions (e.g., 3.11, 3.12) | Ongoing updates with better speed, typing, async features, etc. |

1. Advantages of using Python over other programming languages.

And. print("Hello, world!") # One line, no semicolons, no main()

**Less Code, More Power**

* Tasks that take many lines in C++ or Java can be done in fewer lines in Python.
* Helps developers be more **productive**.

**Interpreted Language**

* Python does **not need to be compiled**.
* Errors are shown at runtime, making debugging easier.

**Cross-Platform Support**

* Python code runs on **Windows**, **Linux**, **macOS**, etc.
* Write once, run anywhere (with Python installed).

**Huge Standard Library**

* Comes with built-in modules for math, file handling, web, JSON, etc.
* No need to install external tools for many common tasks.

**Supports Multiple Programming Paradigms**

* Python supports:
  + **Object-Oriented Programming (OOP)**
  + **Procedural programming**
  + **Functional programming**

**Popular for Emerging Fields**

* Python is a top choice for:
  + **Data Science**
  + **Artificial Intelligence (AI)**
  + **Machine Learning (ML)**
  + **Web Development**
  + **Automation**

**Large Community and Support**

* Millions of developers use Python.
* Tons of tutorials, forums (like Stack Overflow), and open-source tools are available.

**Rich Ecosystem of Libraries and Frameworks**

* Examples:
  + **Web Development** – Django, Flask
  + **Data Science** – NumPy, Pandas, Matplotlib
  + **Machine Learning** – TensorFlow, scikit-learn

**Free and Open Source**

* Python is completely free to download and use.
* Open-source license allows modification and contribution.

1. Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).

#### Ans. ✅ For Windows / Mac / Linux:

1. Go to the official Python website:  
   👉 https://www.python.org/downloads/
2. Click **“Download Python [version]”**
3. Run the installer:
   * **✅ Tick "Add Python to PATH"**
   * Click **Install Now**

### 🔍 ****Verify Installation****

Open **Command Prompt (Windows)** or **Terminal (Mac/Linux)** and type:

bash

CopyEdit

python --version

You should see the installed Python version.

## ⚙️ **Option 1: Using Anaconda (Recommended for Data Science)**

### 📦 What is Anaconda?

Anaconda is a distribution of Python with pre-installed packages for **data science, ML, and statistics** (like Pandas, NumPy, Jupyter, etc.).

### ✅ Steps:

1. Download from:  
   👉 https://www.anaconda.com/products/distribution
2. Run the installer and follow the setup.
3. Open **Anaconda Navigator** or **Jupyter Notebook** from the start menu.

## 🖥️ **Option 2: Using PyCharm (Good for Beginners & Projects)**

### 🧰 What is PyCharm?

PyCharm is a popular **Python IDE** (Integrated Development Environment) from JetBrains, useful for writing, debugging, and running Python projects.

### ✅ Steps:

Go to:  
👉 https://www.jetbrains.com/pycharm/download

Download the **Community Edition** (free).

Install and open PyCharm.

6.

Ans. # This is a simple Python program

print("Hello, World!")

7. Set up Python on your local machine and write a program to display your name.

* Ans. Go to: <https://www.python.org>
* Click **Download Python**
* Run the file and click **Install Now**
* ✅ Make sure you **tick "Add Python to PATH"** during installation.

print("My name is Abdul R.")

8. Understanding Python’s PEP 8 guidelines.

### Ans. 1. ****Indentation****

* Use **4 spaces** per indentation level.
* No tabs (only spaces).

### 2. ****Maximum Line Length****

* Keep each line **less than 79 characters**.
* For long comments or docstrings, use **72 characters**.

### 3. ****Blank Lines****

* Use blank lines to **separate functions, classes**, and blocks of code.
* Improves readability.

### 4. ****Imports****

* Put all import statements at the **top of the file**.
* Import **one module per line**.

### 5. ****Naming Conventions****

* **Variables, functions** → snake\_case (e.g., user\_name)
* **Constants** → ALL\_CAPS (e.g., MAX\_SIZE)
* **Classes** → CamelCase (e.g., StudentDetails)

### 6. ****Spaces in Expressions****

* Use **spaces** around operators and after commas.
* Example: x = a + b (not x=a+b)

### 7. ****Comments****

* Write comments to explain **why** something is done.
* Use full sentences and start with a **capital letter**.

### 8. ****Docstrings****

* Use triple quotes """ """ for writing **documentation strings** for functions, classes, or modules.

## 🧰 **Tools to Help Follow PEP 8**

* **flake8** – checks code style
* **pylint** – finds errors and style issues
* **black** – automatically formats code to match PEP 8

9. Indentation, comments, and naming conventions in Python.

### Ans. . ****Indentation****

#### ➤ **Definition:**

Indentation means adding **spaces at the beginning of a line** of code.  
In Python, indentation is **not optional**—it is **required** to show which lines belong to the same block of code.

#### ➤ **Purpose:**

Indentation is used to define **blocks of code**, such as those inside:

* functions
* loops
* conditionals (if-else)
* classes

#### ➤ **Standard Rule (PEP 8):**

* Use **4 spaces** for each indentation level.
* Do **not mix tabs and spaces**.

#### ➤ **Example:**

python

CopyEdit

def greet():

print("Hello, World!")

If indentation is not used correctly, Python will give an **IndentationError**.

### ✅ 2. ****Comments****

#### ➤ **Definition:**

Comments are lines in the code that are **not executed**.  
They are written to explain what the code does, making it easier to read and understand.

#### ➤ **Types of Comments:**

* **Single-line comment:** Starts with #
* **Multi-line comment:** Enclosed in triple quotes ''' or """ (also used for documentation)

#### ➤ **Purpose of Comments:**

* Explain **what the code does**
* Improve **readability**
* Help others (and your future self) understand the logic

#### ➤ **Examples:**

python

CopyEdit

# This is a single-line comment

'''

This is a

multi-line comment

'''

### ✅ 3. ****Naming Conventions****

#### ➤ **Definition:**

Naming conventions refer to **standard rules** for naming **variables, functions, classes, and constants** in Python.

#### ➤ **Why It Matters:**

* Makes code **clean, consistent, and readable**
* Helps avoid **confusion and errors**

10. Writing readable and maintainable code.

### Ans. Use Proper Indentation

* Indent code using **4 spaces**.
* Clearly shows the **structure and logic** of your code.

python

CopyEdit

def greet(name):

print("Hello", name)

**2. 💬 Write Clear Comments**

* Use # to add comments that explain **why** something is done.
* Avoid obvious comments (don’t explain what is already clear).

python

CopyEdit

# Calculate total price after discount

total = price - discount

**3. 🧾 Follow Naming Conventions**

Use meaningful names:

* Variables/functions: snake\_case
* Classes: CamelCase
* Constants: ALL\_CAPS

Bad: a = 5  
Good: student\_age = 5

**4. 📦 Keep Code Organized**

* Divide your program into **functions** or **modules**.
* Each function should do **one task only**.

python

CopyEdit

def calculate\_area(length, width):

return length \* width

**5. 🧹 Avoid Repetition (DRY Principle)**

**D.R.Y. = Don’t Repeat Yourself**  
Reuse functions instead of repeating code.

Bad:

python

CopyEdit

print(a \* a)

print(b \* b)

Good:

python

CopyEdit

def square(x):

return x \* x

**6. 🧪 Handle Errors Gracefully**

Use try-except blocks to manage errors.

python

CopyEdit

try:

age = int(input("Enter age: "))

except ValueError:

print("Please enter a valid number.")

**7. 📑 Use Docstrings**

Add a description at the start of your functions using triple quotes.

python

CopyEdit

def greet(name):

"""This function prints a greeting message."""

print("Hello", name)

**8. 🧰 Use Tools for Clean Code**

* Format code with tools like black
* Check code with flake8 or pylint

11. Write a Python program that demonstrates the correct use of indentation, comments, and variables following PEP 8 guidelines.

Ans. # Program to calculate the area of a rectangle

def calculate\_area(length, width):

"""

This function calculates and returns

the area of a rectangle.

"""

area = length \* width # Multiplying length and width

return area

def main():

# Input values from the user

length = float(input("Enter the length of the rectangle: "))

width = float(input("Enter the width of the rectangle: "))

# Call the function and display the result

rectangle\_area = calculate\_area(length, width)

print("The area of the rectangle is:", rectangle\_area)

# Call the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

12. Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

## Ans. 1. **Integer (**int**)**

### ➤ Used to store ****whole numbers**** (positive or negative)

Examples: -5, 0, 99

python

CopyEdit

age = 25

## 🔣 2. **Float (**float**)**

### ➤ Used to store ****decimal (floating-point) numbers****

Examples: 3.14, -0.5, 100.0

python

CopyEdit

price = 19.99

## 🔤 3. **String (**str**)**

### ➤ A ****sequence of characters**** (letters, digits, symbols) enclosed in quotes

Examples: "Hello", 'Python'

python

CopyEdit

name = "Abdul"

## 📚 4. **List (**list**)**

### ➤ An ****ordered, changeable**** collection of items

* Items can be of **any data type**
* Defined using **square brackets** []

python

CopyEdit

fruits = ["apple", "banana", "mango"]

fruits[0] # Output: "apple"

✅ You can **add, remove, or change** items in a list.

## 📦 5. **Tuple (**tuple**)**

### ➤ An ****ordered, unchangeable**** collection of items

* Defined using **round brackets** ()

python

CopyEdit

coordinates = (10, 20)

coordinates[1] # Output: 20

❌ You **cannot modify** a tuple once it's created.

## 🗂️ 6. **Dictionary (**dict**)**

### ➤ A ****collection of key-value pairs****

* Defined using **curly braces** {}
* Each item has a **key** and a **value**

python

CopyEdit

student = {"name": "Abdul", "age": 20}

student["name"] # Output: "Abdul"

✅ You can **add, change, or delete** key-value pairs.

## 🔘 7. **Set (**set**)**

### ➤ An ****unordered collection of unique items****

* No duplicate values allowed
* Defined using curly braces {}

python

CopyEdit

numbers = {1, 2, 3, 2}

print(numbers) # Output: {1, 2, 3}

13. Python variables and memory allocation.

## Ans. 1. **What is a Variable in Python?**

A **variable** in Python is a **name** that refers to a value stored in memory.  
It acts like a **label** or **container** that holds data.

### 🧾 Example:

python

CopyEdit

x = 10

name = "Abdul"

* x and name are variables.
* 10 and "Abdul" are values stored in memory.

## ✅ 2. **How Variables Work in Python**

* In Python, **everything is an object**.
* When you assign a value to a variable:
  + Python creates an **object** in memory.
  + The variable **points to** that object.

### 🔁 Example:

python

CopyEdit

a = 5

b = a

* Both a and b point to the **same object** (5) in memory.

## ✅ 3. **Memory Allocation in Python**

When you assign a value to a variable:

1. Python creates an object in **RAM (memory)**.
2. The object has:
   * A **type** (e.g., int, str)
   * A **value** (e.g., 10, "Hello")
   * An **ID (memory address)**

### 🧪 Example:

python

CopyEdit

x = 100

print(id(x)) # Shows the memory address of the object

## ✅ 4. **Dynamic Typing in Python**

* Python is **dynamically typed**, meaning:
  + You **don’t need to declare** the data type.
  + A variable can **change type** during execution.

### Example:

python

CopyEdit

x = 10 # x is an integer

x = "hello" # now x is a string

## ✅ 5. **Garbage Collection**

* Python has a built-in **garbage collector**.
* When a variable is **no longer used**, memory is **automatically freed**.
* Python uses **reference counting** to track how many variables point to an object.

14. Python operators: arithmetic, comparison, logical, bitwise.

## Ans. **Arithmetic Operators**

Used to perform **mathematical operations**.

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| + | Addition | 10 + 5 | 15 |
| - | Subtraction | 10 - 5 | 5 |
| \* | Multiplication | 10 \* 5 | 50 |
| / | Division | 10 / 5 | 2.0 |
| // | Floor Division | 10 // 3 | 3 |
| % | Modulus (remainder) | 10 % 3 | 1 |
| \*\* | Exponent (power) | 2 \*\* 3 | 8 |

## 2️⃣ **Comparison Operators**

Used to **compare values**. Result is always True or False.

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal to | 5 == 5 | True |
| != | Not equal to | 5 != 3 | True |
| > | Greater than | 5 > 3 | True |
| < | Less than | 5 < 3 | False |
| >= | Greater than or equal to | 5 >= 5 | True |
| <= | Less than or equal to | 5 <= 2 | False |

## 3️⃣ **Logical Operators**

Used to combine **conditional statements**.

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| and | True if **both** are True | 5 > 3 and 6 > 2 | True |
| or | True if **one** is True | 5 > 3 or 2 > 6 | True |
| not | Reverses the result | not(5 > 3) | False |

## 4️⃣ **Bitwise Operators**

Operate on **binary (bit-level)** data.

| **Operator** | **Meaning** | **Example** | **Binary Operation** | **Result** |
| --- | --- | --- | --- | --- |
| & | AND | 5 & 3 | 0101 & 0011 = 0001 | 1 |
| ` | ` | OR | `5 | 3` |
| ^ | XOR (exclusive OR) | 5 ^ 3 | 0101 ^ 0011 = 0110 | 6 |
| ~ | NOT (1's complement) | ~5 | ~0101 = 1010 | -6 |
| << | Left shift | 5 << 1 | 0101 → 1010 | 10 |
| >> | Right shift | 5 >> 1 | 0101 → 0010 | 2 |

## ✅ Summary Table:

14. Write a Python program to demonstrate the creation of variables and different data types.

Ans. # Demonstrating variables and different data types in Python

# Integer

age = 25

print("Age:", age)

print("Data type of 'age':", type(age))

# Float

price = 19.99

print("\nPrice:", price)

print("Data type of 'price':", type(price))

# String

name = "Abdul"

print("\nName:", name)

print("Data type of 'name':", type(name))

# Boolean

is\_student = True

print("\nIs student:", is\_student)

print("Data type of 'is\_student':", type(is\_student))

# List

fruits = ["apple", "banana", "cherry"]

print("\nFruits:", fruits)

print("Data type of 'fruits':", type(fruits))

# Tuple

coordinates = (10, 20)

print("\nCoordinates:", coordinates)

print("Data type of 'coordinates':", type(coordinates))

# Dictionary

student = {"name": "Abdul", "age": 20, "course": "Python"}

print("\nStudent Info:", student)

print("Data type of 'student':", type(student))

# Set

unique\_number\_

15. Practical Example 1: How does the Python code structure work?

Ans. # Importing necessary module

from datetime import datetime

# Function to calculate birth year

def calculate\_birth\_year(age):

current\_year = datetime.now().year

birth\_year = current\_year - age

return birth\_year

# Main function

def main():

# Input from the user

name = input("Enter your name: ")

age = int(input("Enter your age: "))

# Calculate birth year using function

birth\_year = calculate\_birth\_year(age)

# Display the result

print(f"Hello, {name}! You were probably born in {birth\_year}.")

# Calling the main function to start the program

if \_\_name\_\_ == "\_\_main\_\_":

main()

16. Practical Example 2: How to create variables in Python?

And. # Creating variables to store student information

student\_name = "Abdul" # string variable

student\_age = 20 # integer variable

student\_grade = 85.5 # float variable

# Displaying the values

print("Student Name:", student\_name)

print("Student Age:", student\_age)

print("Student Grade:", student\_grade)

17. Practical Example 3: How to take user input using the input() function.

Ans. # Taking user input using input() function

name = input("Enter your name: ") # Takes input as string

age = input("Enter your age: ") # Takes input as string by default

# Display the information

print("Hello", name + "!")

print("You”)

17. Practical Example 4: How to check the type of a variable dynamically using type().

And. # Taking input from the user

name = input("Enter your name: ") # string input

age = int(input("Enter your age: ")) # integer input

marks = float(input("Enter your marks: ")) # float input

# Displaying the values and their types

print("\n--- Variable Types ---")

print("Name:", name, "| Type:", type(name))

print("Age:", age, "| Type:", type(age))

print("Marks:", marks, "| Type:", type(marks))

18. Introduction to conditional statements: if, else, elif.

Ans. age = 16

if age >= 18:

print("You are eligible to vote.")

else:

print("You are not eligible to vote.")

19. Nested if-else conditions.

And. num = int(input("Enter a number: "))

if num > 0:

print("The number is positive.")

if num % 2 == 0:

print("It is even.")

else:

print("It is odd.")

else:

print("The number is not positive.")

20. Practical Example 5: Write a Python program to find greater and less than a number using if\_else.

Ans. # Program to compare two numbers using if-else

# Taking input from the user

num1 = int(input("Enter first number: "))

num2 = int(input("Enter second number: "))

# Comparing numbers

if num1 > num2:

print(num1, "is greater than", num2)

elif num1 < num2:

print(num1, "is less than", num2)

else:

print("Both numbers are equal.")

21. Practical Example 6: Write a Python program to check if a number is prime using if\_else.

Ans. # Program to check if a number is prime

num = int(input("Enter a number: "))

if num <= 1:

print(num, "is not a prime number.")

else:

is\_prime = True # Assume the number is prime

# Check for factors

for i in range(2, int(num\*\*0.5) + 1):

if num % i == 0:

is\_prime = False

break

# Result based on the flag

if is\_prime:

print(num, "is a prime number.")

else:

print(num, "is not a prime number.")

22. Practical Example 7: Write a Python program to calculate grades based on percentage using if-else ladder.

Ans. # Program to calculate grade based on percentage

# Taking input from the user

percentage = float(input("Enter your percentage: "))

# Using if-else ladder to determine grade

if percentage >= 90 and percentage <= 100:

print("Grade: A")

elif percentage >= 80:

print("Grade: B")

elif percentage >= 70:

print("Grade: C")

elif percentage >= 60:

print("Grade: D")

elif percentage >= 50:

print("Grade: E")

elif percentage >= 0:

print("Grade: F")

else:

print("Invalid percentage!")

23. Practical Example 8: Write a Python program to check if a person is eligible to donate blood using a nested if

Ans. # Program to check blood donation eligibility using nested if

# Input from the user

age = int(input("Enter your age: "))

if age >= 18:

weight = float(input("Enter your weight (in kg): "))

if weight >= 50:

print("You are eligible to donate blood.")

else:

print("You are not eligible to donate blood due to low weight.")

else:

print("You are not eligible to donate blood due to age.")

24. Introduction to for and while loops.

Ans. for variable in sequence:

# code block

for i in range(1, 6):

print(i)

while condition:

# code block

25. How loops work in Python.

1. Ans. A **sequence** (like a list or a range) is provided.
2. The loop runs **once for each item** in the sequence.
3. The variable inside the for loop takes the **value of each item**, one by one.

#### 🧾 Example:

python

CopyEdit

for i in range(1, 4):

print("Hello", i)

🧠 Working:

* Iteration 1: i = 1 → prints Hello 1
* Iteration 2: i = 2 → prints Hello 2
* Iteration 3: i = 3 → prints Hello 3

### ✅ ****2.**** while ****Loop (Indefinite Loop)****

Used when you **don’t know how many times** the loop will run — it continues **as long as the condition is true**.

#### 💡 How it works:

1. The condition is checked.
2. If **true**, the loop body executes.
3. After each loop, the condition is checked again.
4. The loop stops when the condition becomes **false**.

#### 🧾 Example:

python

CopyEdit

i = 1

while i <= 3:

print("Hello", i)

i += 1

🧠 Working:

* Check: i = 1, condition True → print Hello 1, then i = 2
* Check: i = 2, condition True → print Hello 2, then i = 3
* Check: i = 3, condition True → print Hello 3, then i = 4
* Check: i = 4, condition False → loop ends

26. Using loops with collections (lists, tuples, etc.).

Ans. fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

📌 Output:

nginx

Copy

Edit

apple

banana

cherry

✅ 2. Looping Through a Tuple

python

Copy

Edit

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

for color in colors:

print(color)

📌 Output:

Copy

Edit

red

green

blue

27. Practical Example 1: Write a Python program to print each fruit in a list using a simple for loop. List1 = ['apple', 'banana', 'mango']

Ans. # Define the list

List1 = ['apple', 'banana', 'mango']

# Use a for loop to print each fruit

for fruit in List1:

print(fruit)

28. Practical Example 2: Write a Python program to find the length of each string in List1.

Ans . # Practical Example 2: Find the length of each string in List1

List1 = ['apple', 'banana', 'mango']

for fruit in List1:

print(f"The length of '{fruit}' is {len(fruit)}")

29. Practical Example 3: Write a Python program to find a specific string in the list using a simple for loop and if condition.

Ans. # Practical Example 3: Find a specific string in the list

List1 = ['apple', 'banana', 'mango']

search = input("Enter the fruit to search: ")

found = False # flag to check if found

for fruit in List1:

if fruit == search:

print(f"'{search}' is found in the list.")

found = True

break

if not found:

print(f"'{search}' is not found in the list.")

30. Practical Example 4: Print this pattern using nested for loop:

Ans. # Practical Example 4: Print a right-angled triangle pattern

rows = 5

for i in range(1, rows + 1):

for j in range(1, i + 1):

print("\*", end=" ")

print() # move to next line after inner loop

31. Understanding how generators work in Python.

Ans. def my\_generator():

yield 1

yield 2

yield 3

gen = my\_generator()

for value in gen:

print(value)

32. Difference between yield and return.

Ans def get\_numbers():

return [1, 2, 3]

print(get\_numbers())

33. Understanding iterators and creating custom iterators.

Ans. my\_list = [10, 20, 30]

it = iter(my\_list) # get iterator object

print(next(it)) # 10

print(next(it)) # 20

print(next(it)) # 30

# print(next(it)) # Raises StopIteration

34. Write a generator function that generates the first 10 even numbers.

Ans. def generate\_even\_numbers():

num = 2

count = 0

while count < 10:

yield num

num += 2

count += 1

35. Write a Python program that uses a custom iterator to iterate over a list of integers.

Ans. class ListIterator:

def \_\_init\_\_(self, data):

self.data = data

self.index = 0

def \_\_iter\_\_(self):

return self # returns the iterator object itself

def \_\_next\_\_(self):

if self.index < len(self.data):

value = self.data[self.index]

self.index += 1

return value

else:

raise StopIteration # signals the end of iteration

36. Defining and calling functions in Python.

Ans. def greet(name="User"):

print("Hello,", name)

greet() # uses default

greet("Abdul") # overrides default

37. Function arguments (positional, keyword, default).

Ans. def show\_info(name, age=18, city="Mumbai"):

print(f"Name: {name}, Age: {age}, City: {city}")

# Positional + Default

show\_info("Sara")

# Positional + Keyword

show\_info("Ali", city="Delhi")

# All Keyword

show\_info(name="Zara", age=30, city="Chennai")

38. Scope of variables in Python.

## Ans. ypes of Scope in Python

| **Type** | **Description** |
| --- | --- |
| **Local** | Inside a function or block |
| **Global** | Outside all functions |
| **Enclosed** | In a nested function (outer function) |
| **Built-in** | Reserved names in Python (like len, print) |

39. Built-in methods for strings, lists, etc.

Ans. s = "hello world"

fruits = ['apple', 'banana', 'mango']

num = -5.7

student = {'name': 'John', 'age': 20}

40. Practical Example: 1) Write a Python program to print "Hello" using a string.

Ans. # Practical Example 1: Print "Hello" using a string

message = "Hello"

print(message)

41. Practical Example: 2) Write a Python program to allocate a string to a variable and print it.

Ans. # Practical Example 2: Allocate a string to a variable and print it

greeting = "Welcome to Python programming!"

print(greeting)

42. Practical Example: 3) Write a Python program to print a string using triple quotes.

Ans. # Practical Example 3: Print a string using triple quotes

message = """This is a string

that spans multiple lines,

using triple quotes."""

print(message)

43. Practical Example: 4) Write a Python program to access the first character of a string using index value.

Ans. # Practical Example 4: Access the first character of a string using index

text = "Python"

first\_char = text[0] # Indexing starts from 0

print("The first character is:", first\_char)

44. Practical Example: 5) Write a Python program to access the string from the second position onwards using slicing.

Ans. # Practical Example 5: Access the string from the second position onwards using slicing

text = "Python"

sliced\_text = text[1:] # Starts from index 1 (second character) to the end

print("Sliced string from second position:", sliced\_text)

46. Practical Example: 6) Write a Python program to access a string up to the fifth character.

Ans. # Practical Example 6: Access a string up to the fifth character

text = "PythonProgramming"

sliced\_text = text[:5] # Slices from index 0 up to (but not including) index 5

print("String up to the fifth character:", sliced\_text)

47. Practical Example: 7) Write a Python program to print the substring between index values 1 and 4.

Ans. # Practical Example 7: Print the substring between index 1 and 4

text = "Python"

substring = text[1:4] # Slices from index 1 to 3 (4 is not included)

print("Substring from index 1 to 4:", substring)

48. Practical Example: 8) Write a Python program to print a string from the last character.

Ans. # Practical Example 8: Print the last character of a string

text = "Python"

last\_char = text[-1] # Negative index -1 refers to the last character

print("The last character is:", last\_char)

49. Practical Example: 9) Write a Python program to print every alternate character from the string starting from index 1.

Ans. # Practical Example 9: Print every alternate character from index 1

text = "PythonProgramming"

result = text[1::2] # Start at index 1, step by 2

print("Alternate characters from index 1:", result)

50. Understanding the role of break, continue, and pass in Python loops.

Ans. for i in range(1, 10):

if i == 5:

break

print(i)

51. Practical Example: 1) Write a Python program to skip 'banana' in a list using the continue statement. List1 = ['apple', 'banana', 'mango']

Ans. # Practical Example 1: Skip 'banana' using continue

List1 = ['apple', 'banana', 'mango']

for fruit in List1:

if fruit == 'banana':

continue # Skip 'banana'

print(fruit)

52. Practical Example: 2) Write a Python program to stop the loop once 'banana' is found using the break statement.

Ans. # Practical Example 2: Stop the loop once 'banana' is found using break

List1 = ['apple', 'banana', 'mango', 'orange']

for fruit in List1:

if fruit == 'banana':

break # Exit the loop when 'banana' is found

print(fruit)

53. Understanding how to access and manipulate strings.

Ans.

| **Task** | **Syntax / Example** |
| --- | --- |
| Access by index | s[0], s[-1] |
| Slice a string | s[1:4], s[:5], s[::2] |
| Change case | s.upper(), s.lower() |
| Replace characters | s.replace('a', 'b') |
| Split into words | s.split() |
| Check substring | 'a' in s, s.find('a') |

54. Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).

Ans.

str1 = "Hello"

str2 = "World"

result = str1 + " " + str2

print(result)

55. String slicing

Ans. string[start : stop : step]

start: index where slicing begins (inclusive)

stop: index where slicing ends (exclusive)

step: how many characters to skip (default is 1

56. Write a Python program to demonstrate string slicing.

Ans. # String slicing demonstration

text = "PythonProgramming"

print("Original string:", text)

# 1. Slice from index 0 to 5 (first 6 characters)

print("text[0:6] =", text[0:6])

# 2. Slice from index 6 to the end

print("text[6:] =", text[6:])

# 3. Slice from start to index 5

print("tex

57. Write a Python program that manipulates and prints strings using various string methods.

Ans. # Define a sample string

text = " python programming is FUN! "

# 1. Print original string

print("Original string:", repr(text))

# 2. Remove leading/trailing whitespace

print("strip():", text.strip())

# 3. Convert to uppercase

print("upper():", text.upper())

58. How functional programming works in Python

### Ans. Key Concepts in Functional Programming:

| **Concept** | **Description** |
| --- | --- |
| **Pure functions** | Functions that always return the same output for the same input and have no side effects |
| **Immutability** | Avoid modifying data — use new data instead |
| **First-class functions** | Functions are treated like variables — passed as arguments, returned from other functions |
| **Higher-order functions** | Functions that take other functions as arguments or return them |
| **Lambda functions** | Anonymous functions written using the lambda keyword |
| **Map, Filter, Reduce** | Built-in functional tools for processing iterables |

59. Using map(), reduce(), and filter() functions for processing data.

Ans. from functools import reduce

numbers = [1, 2, 3, 4, 5, 6]

# Step 1: Filter even numbers

evens = list(filter(lambda x: x % 2 == 0, numbers))

# Step 2: Square the even numbers

squares = list(map(lambda x: x \*\* 2, evens))

# Step 3: Sum the squared numbers

sum\_of\_squares = reduce(lambda x, y: x + y, squares)

print("Filtered evens:", evens)

print("Squared evens:", squares)

print("Sum of squared evens:", sum\_of\_squares)

60. Write a Python program to apply the map() function to square a list of numbers.

Ans. # Define a list of numbers

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

# Use map() with a lambda function to square each number

squared\_numbers = list(map(lambda x: x \*\* 2, numbers))

# Print the result

print("Original numbers:", numbers)

print("Squared numbers:", squared\_numbers)

61. Write a Python program that uses reduce() to find the product of a list of numbers.

Ans. from functools import reduce

# Define a list of numbers

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

# Use reduce() to multiply all numbers in the list

product = reduce(lambda x, y: x \* y, numbers)

# Print the result

print("Original numbers:", numbers)

print("Product of numbers:", product)

62. Write a Python program that filters out even numbers using the filter() function.

Ans. # Define a list of numbers

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Use filter() to keep only even numbers

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

# Print the result

print("Original numbers:", numbers)

print("Even numbers:", even\_numbers)

# Function to calculate grade based on marks

def calculate\_grade(marks):

if marks >= 90:

return 'A+'

elif marks >= 80:

return 'A'

elif marks >= 70:

return 'B'

elif marks >= 60:

return 'C'

elif marks >= 50:

return 'D'

else:

return 'F'

# Function to display all student records

def display\_students(students):

if not students:

print("No student records found.")

return

print("\n--- Student Records ---")

for student in students:

print(f"Name: {student['name']}, Marks: {student['marks']}, Grade: {student['grade']}")

print()

# Main program loop

def grade\_management\_system():

students = []

while True:

print("\n==== Grade Management Menu ====")

print("1. Add Student")

print("2. View All Students")

print("3. Exit")

choice = input("Enter your choice (1-3): ")

if choice == '1':

name = input("Enter student name: ")

try:

marks = float(input("Enter marks (0-100): "))

if marks < 0 or marks > 100:

print("❌ Marks must be between 0 and 100.")

continue

except ValueError:

print("❌ Invalid input for marks.")

continue

grade = calculate\_grade(marks)

students.append({'name': name, 'marks': marks, 'grade': grade})

print(f"✅ Student '{name}' added with grade '{grade}'.")

elif choice == '2':

display\_students(students)

elif choice == '3':

print("👋 Exiting Grade Management System.")

break

else:

print("❌ Invalid choice. Please select 1, 2, or 3.")

# Run the application

grade\_management\_system()