ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION)

WHY PYTHON? – FEATURES & APPLICATIONS

CHAPTER 1: INTRODUCTION TO PYTHON

PYTHON IS ONE OF THE MOST **POPULAR AND VERSATILE PROGRAMMING LANGUAGES** IN THE WORLD. IT IS WIDELY USED IN VARIOUS FIELDS, INCLUDING **WEB DEVELOPMENT, ARTIFICIAL INTELLIGENCE, DATA SCIENCE, AUTOMATION, AND GAME DEVELOPMENT**.

PYTHON IS KNOWN FOR ITS **SIMPLICITY, READABILITY, AND FLEXIBILITY,** MAKING IT AN EXCELLENT CHOICE FOR BOTH **BEGINNERS AND EXPERIENCED DEVELOPERS**.

📌 **FUN FACT:** PYTHON WAS CREATED BY **GUIDO VAN ROSSUM** IN 1991, AND IT IS NAMED AFTER THE COMEDY SERIES **"MONTY PYTHON"**, NOT THE SNAKE!

CHAPTER 2: WHY PYTHON?

PYTHON IS PREFERRED BY DEVELOPERS AND ORGANIZATIONS FOR SEVERAL REASONS. LET'S EXPLORE ITS **KEY FEATURES** AND **WHY IT STANDS OUT** AMONG OTHER PROGRAMMING LANGUAGES.

2.1 EASY TO LEARN AND USE

- Python has a **simple and readable syntax** that is easy for beginners to understand.
- Unlike other languages like **C++ or Java**, Python code looks more like **plain English**.

📌 **Example:** Printing "Hello, World!" in Python:

```
print("Hello, World!")
```

The same task in **Java** would require multiple lines of code.

2.2 Open-Source and Free

- Python is **completely free** to use and open-source.
- Developers can **modify and distribute** Python without restrictions.

2.3 Cross-Platform Compatibility

- Python works on **Windows, macOS, Linux,** and even mobile devices.
- Write your code once and run it anywhere without modification.

2.4 Large Standard Library

- Python comes with a **rich collection of built-in modules** that simplify tasks like:
  ✓ Working with **files and databases**
  ✓ Sending **emails**
  ✓ Performing **mathematical calculations**

2.5 Supports Object-Oriented and Procedural Programming

- Python allows both **object-oriented programming (OOP)** and **procedural programming,** making it flexible for different projects.

2.6 Extensive Community Support

- Python has a **large global community** of developers.
- You can find **libraries, frameworks, and solutions** for almost any coding challenge.

2.7 High Demand in the Job Market

- Python developers are in **high demand** due to its wide applications in **AI, data science, and software development**.
- Python is used by **top tech companies** like Google, Facebook, Amazon, and Netflix.

Chapter 3: Applications of Python

Python is used in many fields. Let's explore **where Python is commonly applied**.

3.1 Web Development

- Python frameworks like **Django** and **Flask** make it easy to build dynamic websites.
- Many popular websites, including **Instagram and Spotify**, are built using Python.

📌 **Example: Creating a simple web server in Python:**

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def home():
```

```
    RETURN "WELCOME TO PYTHON WEB DEVELOPMENT!"

APP.RUN()
```

3.2 ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

- PYTHON IS THE **LEADING LANGUAGE FOR AI & MACHINE LEARNING** BECAUSE OF LIBRARIES LIKE **TENSORFLOW, PYTORCH, AND SCIKIT-LEARN**.
- AI APPLICATIONS SUCH AS **CHATBOTS, VOICE ASSISTANTS, AND SELF-DRIVING CARS** USE PYTHON.

📌 EXAMPLE: AI-POWERED CHATBOT (BASIC CODE)

```
IMPORT RANDOM

RESPONSES = ["HELLO!", "HOW CAN I HELP YOU?", "NICE TO MEET YOU!", "PYTHON IS AWESOME!"]

USER_INPUT = INPUT("SAY SOMETHING: ")

PRINT(RANDOM.CHOICE(RESPONSES))
```

3.3 DATA SCIENCE & DATA ANALYSIS

- PYTHON IS WIDELY USED IN **DATA SCIENCE AND ANALYTICS** WITH TOOLS LIKE **PANDAS, NUMPY, AND MATPLOTLIB**.
- COMPANIES USE PYTHON FOR **ANALYZING TRENDS, PREDICTING OUTCOMES, AND MAKING BUSINESS DECISIONS**.

📌 EXAMPLE: ANALYZING DATA IN PYTHON

```
IMPORT PANDAS AS PD


DATA = {'NAME': ['ALICE', 'BOB', 'CHARLIE'], 'AGE': [25, 30, 35]}
DF = PD.DATAFRAME(DATA)


PRINT(DF)
```

3.4 Automation & Scripting

- Python can automate **repetitive tasks**, such as file management, email handling, and web scraping.
- It is used in IT operations for **task automation and process efficiency**.

📌 Example: Automating File Renaming

```
IMPORT OS


FOR FILENAME IN OS.LISTDIR("FILES"):
  OS.RENAME(FILENAME, FILENAME.LOWER())
```

3.5 Game Development

- Python is used in gaming with libraries like **Pygame**.
- Games like **Battlefield 2** and **Civilization IV** use Python for scripting.

📌 Example: Creating a Simple Game in Pygame

```
IMPORT PYGAME

PYGAME.INIT()


WIN = PYGAME.DISPLAY.SET_MODE((500, 500))

PYGAME.DISPLAY.SET_CAPTION("SIMPLE GAME")


RUN = TRUE

WHILE RUN:

  FOR EVENT IN PYGAME.EVENT.GET():

    IF EVENT.TYPE == PYGAME.QUIT:

      RUN = FALSE


PYGAME.QUIT()
```

3.6 Cybersecurity & Ethical Hacking

- Python is widely used in **cybersecurity** for **penetration testing, malware analysis, and network security**.
- Ethical hackers use Python to find vulnerabilities in security systems.

📌 Example: Basic Password Generator in Python

```
IMPORT RANDOM

IMPORT STRING
```

```
PASSWORD = ''.JOIN(RANDOM.CHOICES(STRING.ASCII_LETTERS +
STRING.DIGITS, K=8))

PRINT("GENERATED PASSWORD:", PASSWORD)
```

3.7 Internet of Things (IoT) & Robotics

- Python is used in IoT projects to connect **smart devices** like **smart home automation systems**.
- It is also used to program robots using **Raspberry Pi** and **Arduino**.

Chapter 4: Career Opportunities in Python

Python opens doors to many **high-paying career opportunities**.

4.1 Job Roles for Python Developers

✓ **Python Developer** – Builds applications and software.
✓ **AI & Machine Learning Engineer** – Works on AI-driven applications.
✓ **Data Scientist** – Analyzes and interprets large datasets.
✓ **Cybersecurity Analyst** – Protects digital data from cyber threats.
✓ **Game Developer** – Uses Python to create interactive games.

4.2 Freelancing & Remote Work

✓ Python developers can work as **freelancers** and build **websites, automate tasks, and create AI-powered solutions**.
✓ Many Python freelancers earn by creating **automation scripts and AI chatbots**.

4.3 Startup & Entrepreneurship

✓ Many successful **startups use Python** for web apps, AI solutions, and automation tools.
✓ Developers can create **AI-based business tools** or **launch online coding courses**.

📌 **Example:** Dropbox, YouTube, and Reddit were all built using Python!

Chapter 5: Exercise

5.1 Multiple Choice Questions

1. Which company developed Python?
    - (a) Microsoft
    - (b) Google
    - (c) Guido van Rossum
    - (d) Apple
2. Which Python framework is used for web development?
    - (a) TensorFlow
    - (b) Flask
    - (c) NumPy
    - (d) OpenCV
3. What is Python commonly used for?
    - (a) Game development
    - (b) AI & Machine Learning
    - (c) Web Development
    - (d) All of the above

5.2 Practical Task

📌 **Write a Python script that asks for a user's name and prints a greeting.**

```
name = input("Enter your name: ")
print("Welcome, " + name + "!")
```

Chapter 6: Summary

✅ Python is **easy to learn, free, and widely used** in various fields.

✅ It is used in **AI, web development, game design, and automation**.

✅ Python has **high job demand and career opportunities**.

SETTING UP PYTHON & WRITING YOUR FIRST PROGRAM

CHAPTER 1: INTRODUCTION TO PYTHON SETUP

Python is a **high-level, beginner-friendly programming language** used in **web development, data science, artificial intelligence, and automation**. Before you start coding, you need to **install Python** and set up the right tools.

In this chapter, you will learn how to:

✓ Download and install Python on your computer.

✓ Set up a **code editor (IDE)** for writing Python programs.

✓ Write and execute your first Python program.

CHAPTER 2: DOWNLOADING AND INSTALLING PYTHON

2.1 Checking If Python is Already Installed

Before installing Python, check if it is already installed on your system.

- **For Windows:** Open **Command Prompt (cmd)** and type:
- python --version
- **For Mac/Linux:** Open **Terminal** and type:
- python3 --version

If Python is installed, it will show the **Python version number** (e.g., **Python 3.10.4**). If not, follow the installation steps below.

2.2 Installing Python on Windows, Mac, and Linux

For Windows

1. Visit python.org.
2. Click **"Download Python 3.x.x"** (latest version).
3. Open the downloaded file and check **"Add Python to PATH"** before clicking **Install**.
4. Once installed, open **Command Prompt (cmd)** and type:
5. python --version

If Python is installed correctly, it will display the version number.

For Mac

1. Open **Terminal** and type:
2. brew install python

If you don't have **Homebrew**, install it first by typing:

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

For Linux (Ubuntu/Debian)

1. Open **Terminal** and type:
2. sudo apt update
3. sudo apt install python3
4. After installation, check the version:
5. python3 --version

CHAPTER 3: SETTING UP A CODE EDITOR (IDE)

Python programs can be written in:

✓ **IDLE** (Built-in Python editor)

✓ **VS Code** (Visual Studio Code)

✓ **PyCharm** (Popular for advanced projects)

✓ **Jupyter Notebook** (Best for Data Science)

Installing VS Code (Recommended for Beginners)

1. Download **VS Code** from code.visualstudio.com.
2. Install and open VS Code.
3. Go to **Extensions** (Ctrl + Shift + X) and install **Python Extension**.
4. Open a new file, save it as **first_program.py**, and start coding!

CHAPTER 4: WRITING YOUR FIRST PYTHON PROGRAM

Now that Python is installed and set up, let's write and execute your **first Python program**.

4.1 The "Hello, World!" Program

The first program every programmer writes is **Hello, World!**, which prints a simple message on the screen.

Using IDLE or VS Code

1. Open **IDLE or VS Code**.
2. Type the following code:
3. print("Hello, World!")
4. Save the file as **hello.py**.
5. Run the program:

- **In IDLE,** press **F5**.
- **In VS Code,** open **Terminal** and type:
- python hello.py

✓ Output:

Hello, World!

4.2 Understanding the Code

print("Hello, World!")

- print() is a **built-in function** in Python.
- "Hello, World!" is a **string (text)** that will be displayed.

📌 **Python does not require semicolons (;) at the end of lines, unlike Java or C++.**

CHAPTER 5: TAKING USER INPUT IN PYTHON

Python allows users to **enter data** using the input() function.

Example: Ask for the User's Name and Greet Them

name = input("Enter your name: ")

print("Hello, " + name + "! Welcome to Python.")

✓ Output (User enters "Alice")

Enter your name: Alice

Hello, Alice! Welcome to Python.

📌 input() takes user input as a **string**.

CHAPTER 6: RUNNING PYTHON PROGRAMS IN DIFFERENT WAYS

6.1 Running Python Code in Command Line (Terminal)

1. Open **Command Prompt (Windows) or Terminal (Mac/Linux)**.
2. Type python and press **Enter** to enter the Python interactive mode.
3. Type:
4. print("Hello, Python!")
5. Press **Enter,** and Python will execute the command immediately.

6.2 Running Python Scripts from a File

If you save the script as script.py, run it by:

python script.py

✓ The script will execute and display the output in the terminal.

CHAPTER 7: COMMON ERRORS IN PYTHON (AND HOW TO FIX THEM)

Python is beginner-friendly, but errors happen! Here are some common mistakes and their solutions:

Error	Cause	Solution
SyntaxError: EOL while scanning string literal	Missing quotation marks in a string	Ensure quotes "" or '' are complete

IndentationError: expected an indented block	Incorrect indentation	Use **4 spaces or a Tab**
NameError: name 'x' is not defined	Using a variable before defining it	Define the variable before using it
TypeError: can only concatenate str (not "int") to str	Trying to add numbers to strings	Convert numbers using str()

📌 **Tip:** Python errors tell you the exact issue and the line number where it happened!

CHAPTER 8: EXERCISE

8.1 Multiple Choice Questions

1. What is Python?
    - (a) A programming language
    - (b) A snake
    - (c) A database system
    - (d) A web browser
2. Which function is used to display output in Python?
    - (a) print()
    - (b) output()
    - (c) display()
    - (d) write()
3. What is the correct way to take user input in Python?

- (a) input()
- (b) get()
- (c) receive()
- (d) enter()

8.2 Practical Task

📌 **Write a Python program that:**

- Asks for the user's **name and age**.
- Prints a message saying **"Hello [Name], you are [Age] years old!"**.

```
name = input("Enter your name: ")
age = input("Enter your age: ")
print("Hello " + name + ", you are " + age + " years old!")
```

CHAPTER 9: SUMMARY

✅ Python is **easy to install** and runs on **Windows, macOS, and Linux**.
✅ The first program **"Hello, World!"** prints text on the screen using print().
✅ Python can take **user input** using input().
✅ Python programs can be run from **IDLE, VS Code, and Terminal**.
✅ Debugging Python errors helps in **writing better code**.

VARIABLES, DATA TYPES & OPERATORS IN PYTHON

CHAPTER 1: INTRODUCTION TO VARIABLES, DATA TYPES, AND OPERATORS

Python is a **high-level programming language** that uses variables, data types, and operators to perform computations.

- **Variables** store values in memory.
- **Data Types** define the type of data stored (e.g., numbers, text, lists).
- **Operators** perform mathematical, logical, and comparison operations.

Understanding these concepts is **fundamental to writing Python programs** efficiently.

CHAPTER 2: VARIABLES IN PYTHON

2.1 What is a Variable?

A **variable** is a container for storing data values.

📌 **Example of a Variable:**

```
name = "Alice"

age = 25
```

Here,

- name stores a **string** ("Alice").
- age stores a **number** (25).

2.2 Naming Rules for Variables

✓ Must start with a **letter or underscore (_)**.
✓ Cannot start with a **number**.
✓ Can contain **letters, numbers, and underscores**.
✓ **Case-sensitive** (age and Age are different).

📌 **Valid Variable Names:**

```
student_name = "Bob"

_age = 20

score1 = 100
```

📌 **Invalid Variable Names:**
❌ 1name = "Alice" (Cannot start with a number)
❌ student-name = "Bob" (No hyphens allowed)
❌ class = "Python" (class is a reserved keyword)

2.3 Assigning Multiple Variables

Python allows multiple variable assignments in one line.

📌 **Example:**

```
x, y, z = 5, 10, 15

print(x, y, z)
```

✓ **Output:**

```
5 10 15
```

CHAPTER 3: DATA TYPES IN PYTHON

3.1 What Are Data Types?

A **data type** defines the kind of data a variable can store.

3.2 Common Data Types in Python

Data Type	Example	Description
int	x = 10	Whole numbers (positive or negative)
float	y = 3.14	Decimal numbers
str	name = "Alice"	Text (string)
bool	is_active = True	Boolean (True or False)
list	fruits = ["Apple", "Banana"]	Collection of items
tuple	colors = ("Red", "Blue")	Immutable collection
dict	person = {"name": "Bob", "age": 30}	Key-value pairs

3.3 Checking Data Types

Python provides the type() function to check a variable's data type.

📌 **Example:**

```
x = 10
print(type(x))  # Output: <class 'int'>


y = "Hello"
```

```
print(type(y))  # Output: <class 'str'>
```

3.4 Type Conversion (Casting)

Python allows changing a variable's data type.

📌 **Example:** Converting **int to str**

```
age = 25

age_str = str(age)

print("Age is " + age_str)  # Output: Age is 25
```

📌 **Example:** Converting **str to int**

```
num_str = "50"

num = int(num_str)

print(num + 10)  # Output: 60
```

CHAPTER 4: OPERATORS IN PYTHON

4.1 What Are Operators?

Operators are used to perform operations on **variables and values**.

4.2 Types of Operators

Operator Type	Description	Example
Arithmetic Operators	Perform basic math operations	+, -, *, /, %, **, //
Comparison Operators	Compare values	==, !=, >, <, >=, <=

Logical Operators	Combine conditions	and, or, not
Assignment Operators	Assign values to variables	=, +=, -=, *=, /=
Bitwise Operators	Perform bitwise calculations	`&,

4.3 Arithmetic Operators

Arithmetic operators are used for mathematical calculations.

Operator	Description	Example	Output
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	6 * 2	12
/	Division	10 / 2	5.0
//	Floor Division	10 // 3	3
%	Modulus (Remainder)	10 % 3	1
**	Exponentiation	2 ** 3	8

📌 **Example:**

```
a = 10

b = 3

print(a + b)  # Output: 13

print(a // b) # Output: 3
```

4.4 Comparison Operators

Comparison operators **compare values** and return True or False.

Operator	Description	Example	Output
==	Equal to	5 == 5	True
!=	Not equal to	5 != 3	True
>	Greater than	7 > 3	True
<	Less than	4 < 2	False

📌 **Example:**

```
x = 10
y = 20
print(x > y)  # Output: False
```

4.5 Logical Operators

Logical operators **combine multiple conditions**.

Operator	Description	Example	Output
and	Returns True if both conditions are True	True and False	False
or	Returns True if at least one condition is True	True or False	True
not	Reverses the result	not True	False

📌 **Example:**

```
a = 5
b = 10
print(a > 0 and b > 5)  # Output: True
```

CHAPTER 5: EXERCISE

5.1 Multiple Choice Questions

1. What is a correct variable declaration in Python?
    - (a) 1name = "Alice"
    - (b) name = "Alice"
    - (c) name@ = "Alice"
    - (d) name- = "Alice"
2. Which operator is used for exponentiation in Python?
    - (a) **
    - (b) //
    - (c) ^
    - (d) %
3. What is the output of 5 == 5?
    - (a) 5
    - (b) False
    - (c) True
    - (d) None

5.2 Practical Tasks

📌 Task 1: Create a Python program that swaps two variables.

a = 5

```
b = 10

a, b = b, a

print("a =", a, "b =", b)  # Output: a = 10, b = 5
```

📌 **Task 2: Write a Python program that checks if a number is even or odd.**

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

if num % 2 == 0:

   print("Even number")

else:

   print("Odd number")
```

CHAPTER 6: SUMMARY

✅ Variables store data, and Python has multiple **data types**.
✅ Operators **perform calculations and comparisons**.
✅ Logical operators help in **decision-making**.

☑ CONDITIONAL STATEMENTS & LOOPS IN PYTHON

CHAPTER 1: INTRODUCTION TO CONDITIONAL STATEMENTS & LOOPS

In Python, **conditional statements** and **loops** are used to control the flow of a program.

- **Conditional statements** allow the program to make decisions based on conditions (e.g., "If it is raining, take an umbrella").
- **Loops** allow the program to execute a block of code multiple times (e.g., "Repeat 10 times").

Understanding these concepts is essential for writing **efficient and dynamic** Python programs.

CHAPTER 2: CONDITIONAL STATEMENTS IN PYTHON

2.1 What Are Conditional Statements?

Conditional statements help programs **execute different code based on conditions**.
They use **if, elif (else if), and else** statements.

Syntax of Conditional Statements:

```
if condition:
    # Code to execute if the condition is True
elif another_condition:
    # Code to execute if the first condition is False and this is True
```

```
else:
    # Code to execute if all conditions are False
```

2.2 Using if Statement

The **if statement** executes a block of code **only if the condition is True**.

📌 **Example:** Checking if a number is positive.

```
num = 10
if num > 0:
    print("The number is positive")
```

✓ **Output:**

```
The number is positive
```

2.3 Using if-else Statement

The **if-else statement** runs different code based on **True or False** conditions.

📌 **Example:** Checking if a number is positive or negative.

```
num = -5
if num > 0:
    print("Positive number")
else:
    print("Negative number")
```

✓ **Output:**

Negative number

2.4 Using if-elif-else Statement

The **if-elif-else statement** checks multiple conditions.

📌 **Example:** Checking if a number is positive, negative, or zero.

```
num = 0
if num > 0:
    print("Positive number")
elif num < 0:
    print("Negative number")
else:
    print("Zero")
```

✓ **Output:**

Zero

2.5 Nested if Statements

A nested if is an **if statement inside another if statement**.

📌 **Example:** Checking if a person is eligible to vote.

```
age = 20
if age >= 18:
    print("You are eligible to vote")
    if age >= 65:
```

```
    print("You are a senior citizen voter")
else:
  print("You are not eligible to vote yet")
```

✓ Output:

```
You are eligible to vote
```

CHAPTER 3: LOOPS IN PYTHON

3.1 What Are Loops?

Loops **repeat a block of code multiple times**.
Python has two main types of loops:

1. **for loop** – Used when the number of iterations is known.
2. **while loop** – Used when iterations are based on a condition.

3.2 for Loop in Python

The for loop is used to **iterate over a sequence (list, tuple, string, or range)**.

📌 **Example:** Printing numbers from 1 to 5 using for loop.

```
for i in range(1, 6):
  print(i)
```

✓ Output:

```
1
2
3
```

4

5

📌 **Example:** Iterating over a list of names.

```
names = ["Alice", "Bob", "Charlie"]

for name in names:
    print("Hello", name)
```

✓ **Output:**

```
Hello Alice

Hello Bob

Hello Charlie
```

3.3 while Loop in Python

The while loop **runs as long as a condition is True**.

📌 **Example:** Printing numbers from 1 to 5 using while loop.

```
num = 1

while num <= 5:
    print(num)
    num += 1
```

✓ **Output:**

```
1

2

3
```

4

5

3.4 break Statement in Loops

The break statement **exits a loop prematurely** when a condition is met.

📌 **Example:** Stopping the loop when a number reaches 3.

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

✓ **Output:**

1

2

3.5 continue Statement in Loops

The continue statement **skips the current iteration** and moves to the next one.

📌 **Example:** Skipping number 3 in a loop.

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

✓ **Output:**

```
1
2
4
5
```

CHAPTER 4: COMBINING LOOPS AND CONDITIONAL STATEMENTS

Loops and conditionals can be used together for **advanced logic**.

📌 **Example:** Print only even numbers from 1 to 10.

```
for num in range(1, 11):
    if num % 2 == 0:
        print(num)
```

✓ **Output:**

```
2
4
6
8
10
```

📌 **Example:** Find the first number divisible by 5 between 1 and 50.

```
num = 1
while num <= 50:
    if num % 5 == 0:
```

```
    print("Found:", num)
    break
num += 1
```

✓ Output:

Found: 5

CHAPTER 5: EXERCISE

5.1 Multiple Choice Questions

1. What is the correct syntax for an if statement in Python?
    - (a) if x > 5 then:
    - (b) if x > 5:
    - (c) if x > 5 {
    - (d) if (x > 5)
2. What will the following code print?
3. for i in range(1, 4):
4. print(i)
    - (a) 1 2 3
    - (b) 1 2 3 4
    - (c) 2 3 4
    - (d) 1 1 1
5. Which statement will immediately exit a loop?
    - (a) stop

- (b) break
- (c) exit
- (d) continue

5.2 Practical Tasks

📌 **Task 1: Write a Python program that checks if a number is positive, negative, or zero.**

```
num = int(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num < 0:
    print("Negative number")
else:
    print("Zero")
```

📌 **Task 2: Create a Python program that prints numbers from 1 to 20 but skips multiples of 4.**

```
for num in range(1, 21):
    if num % 4 == 0:
        continue
    print(num)
```

📌 **Task 3: Create a guessing game where the user has to guess a number between 1 and 10.**

```
import random
```

```
target = random.randint(1, 10)


while True:
    guess = int(input("Guess a number between 1 and 10: "))
    if guess == target:
        print("Congratulations! You guessed it!")
        break
    else:
        print("Try again!")
```

CHAPTER 6: SUMMARY

✅ **Conditional statements** allow Python to make decisions using if, elif, and else.

✅ **Loops** (for and while) help repeat actions multiple times.

✅ **break exits a loop**, while **continue skips an iteration**.

✅ Combining loops with conditionals helps in solving **complex problems efficiently**.

☑ Basic Functions and User Input in Python

Chapter 1: Introduction to Functions and User Input

In Python, **functions** allow us to write reusable blocks of code that perform specific tasks, while **user input** enables programs to interact with users by receiving data from them.

- **Functions** help in structuring code, making it reusable and efficient.
- **User input** allows programs to take input from users, process it, and generate dynamic outputs.

Understanding these concepts is crucial for developing **interactive and modular** Python applications.

Chapter 2: Basic Functions in Python

2.1 What is a Function?

A function is a block of code that **performs a specific task** when called. Instead of writing the same code multiple times, we can **define a function once and reuse it** whenever needed.

2.2 Why Use Functions?

✓ Avoids code duplication
✓ Improves code readability
✓ Makes debugging easier
✓ Enables modular programming

2.3 Defining and Calling Functions

A function in Python is **defined** using the def keyword.

📌 Syntax of a Function:

```
def function_name():
    # Code inside the function
    print("This is a function")
```

📌 Example: Creating and Calling a Function

```
def greet():
    print("Hello! Welcome to Python Programming.")

greet()  # Calling the function
```

✓ Output:

```
Hello! Welcome to Python Programming.
```

2.4 Function with Parameters

Functions can accept **parameters** (input values) to perform specific tasks.

📌 Example: Function with Parameters

```
def greet_user(name):
    print("Hello, " + name + "!")

greet_user("Alice")  # Calling the function with an argument
```

✓ **Output:**

Hello, Alice!

Here, "Alice" is passed as an argument to the name parameter in the function.

2.5 Function with Return Value

A function can return a value using the return statement.

📌 **Example: Function That Returns a Sum**

```
def add_numbers(a, b):
    return a + b

result = add_numbers(5, 3)
print("Sum:", result)
```

✓ **Output:**

Sum: 8

CHAPTER 3: USER INPUT IN PYTHON

3.1 What is User Input?

User input allows users to **enter data** into a program, making it interactive. In Python, we use the input() function to receive input from the user.

📌 **Example: Taking User Input and Printing It**

```
name = input("Enter your name: ")
```

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

✓ Output (if user enters "John"):

```
Enter your name: John
Hello, John!
```

3.2 Taking Numerical Input

By default, input() returns a **string**. If you need a number, you must **convert the input** using int() or float().

📌 Example: Adding Two Numbers Entered by the User

```
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
sum_result = num1 + num2
print("Sum:", sum_result)
```

✓ Output (if user enters 5 and 7):

```
Enter first number: 5
Enter second number: 7
Sum: 12
```

3.3 Using User Input in Functions

📌 Example: Function That Takes User Input

```
def greet():
    name = input("Enter your name: ")
```

```
    print("Hello, " + name + "! Welcome to Python.")

greet()
```

✓ **Output:**

```
Enter your name: Alice
Hello, Alice! Welcome to Python.
```

CHAPTER 4: COMBINING FUNCTIONS AND USER INPUT

Functions and user input can be combined to make programs more **interactive**.

📌 Example: Function That Multiplies Two User-Entered Numbers

```
def multiply_numbers():
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    result = num1 * num2
    return result

output = multiply_numbers()
print("Multiplication Result:", output)
```

✓ **Output (if user enters 3 and 4):**

```
Enter first number: 3
Enter second number: 4
Multiplication Result: 12.0
```

CHAPTER 5: EXERCISE

5.1 Multiple Choice Questions

1. What is the correct syntax to define a function in Python?
    - (a) function myFunction():
    - (b) def myFunction():
    - (c) myFunction def():
    - (d) create function myFunction():
2. What will be the output of the following code?
3. def my_function():
4. return 10 + 5
5.
6. print(my_function())
    - (a) 15
    - (b) 10
    - (c) 5
    - (d) Error
7. What does the input() function do?
    - (a) Displays output on the screen
    - (b) Receives user input as a string
    - (c) Converts a string to an integer
    - (d) Runs a loop

5.2 Practical Tasks

📌 Task 1: Create a Python function that asks the user for their age and prints whether they are an adult or a minor.

```
def check_age():
    age = int(input("Enter your age: "))
    if age >= 18:
        print("You are an adult.")
    else:
        print("You are a minor.")

check_age()
```

📌 Task 2: Write a function that takes two numbers as user input and prints their sum, difference, and product.

```
def calculate():
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    print("Sum:", num1 + num2)
    print("Difference:", num1 - num2)
    print("Product:", num1 * num2)

calculate()
```

📌 **Task 3: Write a Python function that asks for the user's name and favorite color, then prints a message combining both.**

```
def favorite_color():
    name = input("Enter your name: ")
    color = input("Enter your favorite color: ")
    print(name + " loves the color " + color + "!")

favorite_color()
```

CHAPTER 6: SUMMARY

✅ **Functions** are reusable blocks of code that perform specific tasks.
✅ The def keyword is used to **define functions**.
✅ Functions can **accept parameters** and **return values**.
✅ **User input** is received using the input() function.
✅ Input is **always a string** by default and should be **converted** if needed.
✅ Functions and user input can be combined to **make interactive programs**.

ASSIGNMENT 1:

WRITE A PYTHON PROGRAM TO CHECK IF A NUMBER IS EVEN OR ODD.

💡 Assignment Solution 1: Write a Python Program to Check if a Number is Even or Odd

---

🎯 Objective:

✓ Learn how to take **user input** in Python.
✓ Use **conditional statements (if-else)** to check whether a number is even or odd.
✓ Display the result using the **print()** function.

🛠 Step-by-Step Guide

Step 1: Open Your Python Editor or IDE

- You can use **IDLE, VS Code, Jupyter Notebook, or any Python editor**.
- If you don't have Python installed, run the program in an **online compiler** like Replit or Google Colab.

Step 2: Take User Input

- Use the input() function to ask the user to enter a number.
- Convert the input into an **integer (int)**, because input() always returns a string.

📌 Code:

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

✓ Explanation:

- input("Enter a number: ") prompts the user to enter a number.
- int() converts the entered value from **string to integer** so it can be used in calculations.

Step 3: Use the Modulus Operator (%) to Check Even or Odd

- The **modulus operator (%)** is used to check the remainder when a number is divided by 2.
- If the remainder is 0, the number is **even**; otherwise, it's **odd**.

📌 Code:

```
if num % 2 == 0:
    print("The number is Even.")
else:
    print("The number is Odd.")
```

✓ Explanation:

- num % 2 == 0: If a number is divisible by 2 (remainder is 0), it is **even**.
- else: If the condition is **False**, the number is **odd**.

Step 4: Complete Python Program

Now, combine all the steps into a single Python script.

📌 Final Code:

```
# Python program to check if a number is even or odd
```

```
# Step 1: Take user input
num = int(input("Enter a number: "))

# Step 2: Check if the number is even or odd
if num % 2 == 0:
    print(f"The number {num} is Even.")
else:
    print(f"The number {num} is Odd.")
```

✓ **Explanation:**

- f"The number {num} is Even." uses **f-strings** to format output dynamically.

Step 5: Run and Test Your Program

Test the program by entering different numbers:

✓ **Example 1:**

Enter a number: 8

The number 8 is Even.

✓ **Example 2:**

Enter a number: 5

The number 5 is Odd.

✓ **Example 3:**

Enter a number: 0

The number 0 is Even.

📌 **Note:** 0 is considered an even number because **0 % 2 = 0**.

🔍 Bonus: Handling User Input Errors

To prevent **errors** if a user enters text instead of a number, use a **try-except block**.

📌 Improved Code with Error Handling:

```
try:
    num = int(input("Enter a number: "))
    if num % 2 == 0:
        print(f"The number {num} is Even.")
    else:
        print(f"The number {num} is Odd.")
except ValueError:
    print("Invalid input! Please enter a valid number.")
```

✓ Example (Handling Text Input):

Enter a number: hello

Invalid input! Please enter a valid number.

🔍 Challenge Yourself!

Try modifying the program to:

✅ Check if a number is **positive, negative, or zero** before checking

even/odd.

✅ Allow the user to **check multiple numbers without restarting the program**.

✅ Store the numbers in a **list** and display all results at the end.

🎯 Summary

✅ Used input() to take a number as input.

✅ Used % (modulus operator) to check divisibility by 2.

✅ Used if-else to print "Even" or "Odd".

✅ Improved the program with **error handling**.

ASSIGNMENT 2:

CREATE A PYTHON-BASED CALCULATOR THAT CAN PERFORM BASIC ARITHMETIC OPERATIONS.

Assignment Solution 2: Create a Python-Based Calculator

🎯 Objective:

✓ Learn how to take **user input** for numbers and operations.
✓ Use **conditional statements (if-elif-else)** to perform calculations.
✓ Display results using the **print() function**.
✓ Implement **error handling** to prevent invalid inputs.

🛠 Step-by-Step Guide

Step 1: Open Your Python Editor or IDE

- You can use **IDLE, VS Code, PyCharm, Jupyter Notebook**, or any other Python editor.
- If Python is not installed, use an **online compiler** like Replit or Google Colab.

Step 2: Take User Input for Numbers and Operation

To perform a calculation, the user needs to:

1. **Enter the first number.**
2. **Enter an operator (+, -, , /).*
3. **Enter the second number.**

📌 Code:

```
num1 = float(input("Enter the first number: "))
```

```
operator = input("Enter an operator (+, -, *, /): ")

num2 = float(input("Enter the second number: "))
```

✓ **Explanation:**

- float(input()) ensures that the user can enter **decimal numbers**.
- input() takes the operator as a string (+, -, *, /).

Step 3: Use Conditional Statements to Perform Operations

Now, use **if-elif-else statements** to execute the correct operation based on the user's input.

📌 **Code:**

```
if operator == "+":
    result = num1 + num2
elif operator == "-":
    result = num1 - num2
elif operator == "*":
    result = num1 * num2
elif operator == "/":
    if num2 != 0:  # Prevent division by zero
        result = num1 / num2
    else:
        result = "Error! Division by zero."
else:
```

result = "Invalid operator!"

✓ **Explanation:**

- The program checks the value of operator and performs the corresponding arithmetic operation.
- If the user enters /, the program **checks if num2 is 0** before performing division (to prevent errors).
- If the user enters an invalid operator, the program returns "Invalid operator!".

Step 4: Display the Result

Use the print() function to show the result to the user.

📌 **Code:**

```
print("Result:", result)
```

✓ **Example Outputs:**

Addition Example:

Enter the first number: 10

Enter an operator (+, -, *, /): +

Enter the second number: 5

Result: 15.0

Division by Zero Example:

Enter the first number: 8

Enter an operator (+, -, *, /): /

Enter the second number: 0

Result: Error! Division by zero.

Step 5: Complete Python Program

Now, let's combine all the steps into a **single Python script**.

📌 **Final Code:**

```
# Python-based calculator for basic arithmetic operations

# Step 1: Take user input
num1 = float(input("Enter the first number: "))
operator = input("Enter an operator (+, -, *, /): ")
num2 = float(input("Enter the second number: "))

# Step 2: Perform calculation using conditional statements
if operator == "+":
    result = num1 + num2
elif operator == "-":
    result = num1 - num2
elif operator == "*":
    result = num1 * num2
elif operator == "/":
    if num2 != 0:
        result = num1 / num2
```

```
    else:
        result = "Error! Division by zero."
else:
    result = "Invalid operator!"

# Step 3: Display the result
print("Result:", result)
```

🔍 Bonus: Improving the Calculator with Error Handling

If the user enters **non-numeric values**, the program may crash. To prevent this, use a **try-except block**.

📌 Enhanced Code with Error Handling:

```
try:
    num1 = float(input("Enter the first number: "))
    operator = input("Enter an operator (+, -, *, /): ")
    num2 = float(input("Enter the second number: "))

    if operator == "+":
        result = num1 + num2
    elif operator == "-":
        result = num1 - num2
    elif operator == "*":
        result = num1 * num2
```

```
    elif operator == "/":
        if num2 != 0:
            result = num1 / num2
        else:
            result = "Error! Division by zero."
    else:
        result = "Invalid operator!"

    print("Result:", result)

except ValueError:
    print("Invalid input! Please enter numbers only.")
```

✓ Example Output (Invalid Input Handling):

Enter the first number: hello

Invalid input! Please enter numbers only.

🔍 Challenge Yourself!

Try modifying the calculator to:

✅ **Ask the user if they want to perform another calculation.**

✅ **Include more operations like exponentiation (**) and modulus (%).**

✅ **Use a loop to allow multiple calculations without restarting the program.**

📌 Hint for Looping Version:

```
while True:
    # Take input and perform calculations
    # Ask user if they want to continue or exit
    choice = input("Do you want to perform another calculation? (yes/no): ")
    if choice.lower() != "yes":
        break
```

🎯 Summary

✅ Used input() to take numbers and operator from the user.
✅ Used if-elif-else to perform **addition, subtraction, multiplication, and division**.
✅ Included **division by zero handling** to prevent errors.
✅ Improved the program with **error handling** using try-except.