1. Differentiate between the interpreter and interactive mode in Python with examples.

Python Interpreter Mode:

The interpreter mode in Python executes a full script stored in a .py file.

The Python script is written, saved, and then executed as a complete program.

It is useful for writing large programs.

Example of Interpreter Mode:

```
# File: script.py
print("Hello, Python!")
```

Run this in the terminal:

```
python script.py
```

Output:

```
Hello, Python!
```

Python Interactive Mode (REPL - Read, Evaluate, Print, Loop):

This mode allows executing one statement at a time directly in the Python shell.

Useful for quick testing and debugging.

Example of Interactive Mode:

```
python-repl
>>> print("Hello, Python!")
Hello, Python!
```

Key Differences:

Feature	Interpreter Mode	Interactive Mode
Execution	Executes a .py file	Executes line-by-line
Output	Shows output after execution	Shows output immediately
Use Case	Writing full programs	Quick testing/debugging

2. Describe the different data types available in Python with examples.

Python provides different data types for handling different kinds of values:

1. Numeric Types

Used to store numbers.

Examples:

```
x = 10     # int
y = 3.14   # float
z = 2 + 3j # complex
```

2. Boolean Type

Represents True or False.

Example:

```
is_active = True
print(type(is_active))  # Output: <class 'bool'>
```

3. Sequence Types

str: Represents text data.

list: Ordered, mutable collection.

tuple: Ordered, immutable collection.

Example:

```
name = "Python"
fruits = ["apple", "banana", "cherry"]
coordinates = (10, 20)
```

4. Set Type

Stores unique values and is unordered.

Example:

```
unique_numbers = {1, 2, 3, 4}
```

5. Dictionary Type : Stores data in key-value pairs.

Example:

```
student = {"name": "Alice", "age": 20}
```

3. Explain how input and output operations work in Python.

1. Input Operations (input())

input() function takes user input as a string.

Example:

```
name = input("Enter your name: ")
print("Hello,", name)
```

2. Output Operations (print())

Displays output on the screen.

Example:

```
print("Welcome to Python!")
print("Sum:", 10 + 5)
```

3. Formatted Output

Uses f-strings for better readability.

Example:

```
age = 25
print(f"I am {age} years old.")
```

4. Explain the different types of operators in Python. Explain the working of any two operators.

Python has several types of operators:

1. Arithmetic Operators (+, -, *, /, %, **, //)

2. Comparison Operators (==, !=, >, <, >=, <=)

3. Logical Operators (and, or, not)

4. Assignment Operators (=, +=, -=, *=, /=)

5. Bitwise Operators (&, |, ^, ~, <<, >>)

6. Membership Operators (in, not in)

7. Identity Operators (is, is not)

Example - Arithmetic Operators (+ and //):

```
a = 10
b = 3
print(a + b)   # Output: 13
print(a // b)  # Output: 3 (floor division)
```

5. Construct a Python program that implements a simple number guessing game.

Program Requirements:

The program randomly selects a number between 1 and 100.

The user guesses the number.

The program tells the user if the guess is too high, too low, or correct.

The game continues until the user guesses the correct number or types "exit".

Python Code:

```
import random
number = random.randint(1, 100)
while True:
    guess = input("Guess a number (or type 'exit' to quit): ")
    if guess.lower() == "exit":
        print("Game Over!")
        break
    guess = int(guess)

if guess < number:
        print("Too low!")
    elif guess > number:
        print("Too high!")
    else:
        print("Correct! You guessed it.")
        break
```

6. Construct a recursive Python function that calculates the factorial of a given number n.

Program Requirements:

Uses recursion to calculate the factorial.

Handles cases where n is negative (factorial is not defined for negative numbers).

Python Code:

```
def factorial(n):
    if n < 0:
        return "Invalid input"
    return 1 if n == 0 else n * factorial(n - 1)


print(factorial(5))  # Output: 120
```

7. Develop a Python program that prints all prime numbers between 1 and 100.

Program Requirements:

Uses a for loop to iterate through numbers.

Uses an inner loop to check divisibility.

Uses continue to skip non-prime numbers.

Python Code:

```
for num in range(1, 101):
    if num > 1:
        for i in range(2, num):
            if num % i == 0:
                break
        else:
            print(num, end=" ")
```

8. Python Program for Grade Calculation

University assigns grades based on marks:

Marks	Grade
>=90	A
80-89	B
70-79	C
60-69	D

50-59	E
<50	Fail

Python Code:

```
marks = int(input("Enter marks: "))
if marks >= 90:
    grade = 'A'
elif marks >= 80:
    grade = 'B'
elif marks >= 70:
    grade = 'C'
elif marks >= 60:
    grade = 'D'
elif marks >= 50:
    grade = 'E'
else:
    grade = 'Fail'
print("Grade:", grade)
```

9. Python Program - String Operations

Program Requirements:

Extract first four characters.

Reverse the string using slicing.

Convert the string to uppercase.

Extract a substring from the 3rd to 7th character.

Python Code:

```
s = input("Enter a string: ")
print("First 4 chars:", s[:4])
print("Reversed:", s[::-1])
print("Uppercase:", s.upper())
print("Substring (3-7):", s[3:7])
```