1ST YEAR

COMPUTER SCIENCE

Unit:02 Python Programming

Exercise Short Answer Questions.

1. **Explain the purpose of using comments in Python code?**
   In **Python**, comments are notes written inside the code that are **not executed by the compiler/interpreter**. They are used to explain the code, make it easier to understand, and help others (or yourself) when revisiting the code later.
   Types of Comments in Python:
   1. Single-line Comment
   - **S**tarts with a # (hash symbol).
   - Anything written after # on that line is ignored by Python.

   Example:
   ```
   # This is a single-line comment
   print("Hello, World!")  # This prints a message
   ```
   2. Multi-line Comment
   Python does not have a special syntax for multi-line comments.
   We can use:
   Triple quotes (''' or """) as a docstring-like block (not assigned to anything).
   Example:
   ```
   '''
   This is another way
   to write multi-line comments
   using triple quotes
   '''
   print("Python comments example")
   ```
2. **Describe the difference between integer and float data types in Python. Provide an example of each.**
   1. Integer (int)
   - Whole numbers (positive, negative, or zero).
   - Do not have a decimal point.

   Example values: -10, 0, 25, 1000

   Integer example:
   ```
   x = 25

   y = -7

   print("x:", x)     # Output: 25

   print("y:", y)     # Output: -7
   ```
   2. Float (float)
   - Numbers that have a decimal point.
   - Can also represent numbers in scientific notation (e.g., 1.2e3 means 1200.0).
   - Example values: 3.14, -0.5, 2.0, 1.2e3

Example in Python:
# Float example

```
a = 3.14
print("a:", a)      # Output: 3.14
```

3. **Define operator precedence and give an example of an expression where operator precedence affects the result.**

Operator precedence:
Operator precedence means the order in which math operations happen. In Python, **precedence of operators** means the **order in which operators are evaluated** in an expression. Operators with **higher precedence** are evaluated first.
If two operators have the **same precedence**, then **associativity** (left-to-right or right-to-left) decides the order.

Example:

```
x = 100 / 5 * 2 + 3
print(x)
```

Output: 43.0

Precedence	Operator(s)	Description	Associativity
1	()	Parentheses (grouping), function calls, indexing	Left to Right
2	**	Exponentiation (power)	Right to Left
4	*, /, //, %	Multiplication, Division, Floor division, Modulus	Left to Right
5	+, -	Addition, Subtraction	Left to Right

4. **How does the shorthand if-else statement differ from the regular if-else statement?**

The regular if-else uses multiple lines. Written in **multiple lines**. Clear and easier to understand for beginners.
The shorthand if-else fits into one line (also called ternary operator). Best for simple conditions. Good for compact code. Both do the same thing.

Regular if-else:

```
age = 18
if age >= 18:
   status = "Adult"
else:
   status = "Minor"
```

Shorthand if-else:

```
status = "Adult" if age >= 18 else "Minor"
```

5. **Explain the use of the range() function in a for loop?**

range() function in a for loop:
The range() function is used in a for loop to generate a sequence of numbers. It does not create a list directly but

gives a range object that produces numbers one by one. Mostly used to repeat a loop a specific number of times. range() is used in a for loop to control how many times the loop runs and which numbers it goes through.

Syntax:

```
range (start, stop, step)
start → (optional) starting number (default = 0)
stop → ending number (not included)
step → (optional) difference between numbers (default = 1)
```

Example:

```
for i in range(0, 10, 2):
 print(i)
```

Output: 0 2 4 6 8

6. **Explain how default parameters work in Python functions.**

In Python, default parameters let you set a value that is used if no argument is passed.

Example:

```
def greet(name="Guest"):
   print("Hello", name)

   greet()       # Output: Hello Guest
   greet("Ali")  # Output: Hello Ali
```

If you don't give a name, it uses "Guest" as default.

7. **Explain why modular programming is useful in Python.**

Modular programming:

Modular programming is **dividing a big program into small, independent modules (functions or files)**. A function/module written once can be used again in different programs. Errors can be found easily since each module is separate. If a formula changes, we only update that specific function, not the whole program. Code is organized into logical sections (functions/files). Easier for others (and yourself) to understand. Python allows importing your own modules (import mymodule) or built-in ones (import math).

Example:

```
# Modular program using functions

def area_rectangle(length, width):
   return length * width

def area_circle(radius):
   return 3.14 * radius * radius

# Main program
print("Area of rectangle:", area_rectangle(10, 5))
print("Area of circle:", area_circle(7))
```

8. **Explain the difference between a class and an object in Python.**
Class:

In Python, a **class** is a blueprint or template that defines the structure and behavior of objects. It contains attributes (variables) and methods (functions) which describe what the object will have and what it can do. A class itself does not take up memory; it only provides the design for creating objects.

Object:

An **object**, on the other hand, is an instance of a class. It is the actual entity that is created from the class and stored in memory. Each object has its own unique data, even though it follows the structure defined by the class.

For example:

If we create a class named ***Car*** with attributes like **brand** and **color**, and a **method to drive**, then creating **car1 = Car("Toyota", "Red")** and **car2 = Car("Honda", "Blue")** will give us two different objects. Both are created from the same class but have different values. The class is like the design of a car, while the object is the actual car built from that design.