python qualitative assignment (1-15)

1) Explain arithmetic operations in Python. Illustrate their usage with a simple program.

Arithmetic operations in Python are used to perform basic mathematical calculations on numbers. Python supports several arithmetic operators that work on integer and floating-point values.

Arithmetic Operators in Python

- **Addition (+)** – Adds two values
- **Subtraction (-)** – Subtracts one value from another
- **Multiplication (*)** – Multiplies two values
- **Division (/)** – Divides one value by another and gives float result
- **Floor Division (//)** – Divides and returns integer part
- **Modulus (%)** – Returns remainder
- **Exponentiation (**)** – Raises power

Simple Program

```
a = 10
b = 3

print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
print("Floor Division:", a // b)
print("Modulus:", a % b)
print("Exponent:", a ** b)
```

Output

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.3333333333333335
Floor Division: 3
Modulus: 1
Exponent: 1000
```

2) Explain type conversion in Python by converting user input into integer and float values.

Type conversion in Python means changing one data type into another. User input taken using `input()` is always treated as a **string**, so we must convert it into required data types like `int` or `float`.

Types of Type Conversion

- **Implicit Conversion** – Python automatically converts data type
- **Explicit Conversion** – Programmer manually converts using functions

Common Conversion Functions

- `int()` – converts to integer
- `float()` – converts to float
- `str()` – converts to string

Program Example

```
num1 = input("Enter an integer: ")
num2 = input("Enter a float: ")

num1 = int(num1)
num2 = float(num2)

print("Integer value:", num1)
print("Float value:", num2)
print("Sum:", num1 + num2)
```

Explanation

Here, the user input is converted from string to integer and float using `int()` and `float()` functions.

---

3) Write a Python program to show the use of indentation in conditional statements.

Python uses **indentation instead of brackets** to define blocks of code. Proper indentation is mandatory, especially in conditional statements like `if`, `elif`, and `else`.

Program Example

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

if num > 0:
    print("Number is positive")
    print("This statement is inside if block")
else:
    print("Number is zero or negative")
```

Explanation

- The indented lines belong to the `if` or `else` block.
- Incorrect indentation will cause an error.
- Indentation improves readability and structure.

---

4) Write a Python program to accept two numbers from the user and display their sum, difference, product, and quotient.

This program demonstrates user input, arithmetic operations, and output formatting.

Program

```
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

print("Sum:", a + b)
print("Difference:", a - b)
print("Product:", a * b)

if b != 0:
    print("Quotient:", a / b)
else:
    print("Division by zero not possible")
```

Explanation

- Numbers are converted to float for accurate division.
- Condition prevents division by zero error.

---

5) Differentiate between mutable and immutable datatypes in Python with examples.

In Python, data types are classified as **mutable** and **immutable** based on whether their values can be changed after creation.

Mutable Datatypes

Mutable objects can be modified after creation.

Examples:

- list
- dictionary
- set

```
list1 = [1, 2, 3]
list1[0] = 10
print(list1)
```

Output

```
[10, 2, 3]
```

Immutable Datatypes

Immutable objects cannot be modified once created.

Examples:

- int
- float
- string
- tuple

```
x = 5
x = 10
print(x)
```

Here, a new object is created instead of modifying the old one.

Difference Table

Mutable	Immutable
Can be changed	Cannot be changed
Memory efficient for updates	New object created
Examples: list, dict	Examples: int, tuple

---

6. Differentiate assignment (=) and comparison (==) operators in Python.

The **assignment operator (=)** is used to assign a value to a variable. It stores the value on the right-hand side into the variable on the left-hand side. It does not check any condition; it simply performs storage.

The **comparison operator (==)** is used to compare two values. It checks whether both sides are equal and returns a Boolean result — either `True` or `False`.

Example:

```
x = 5      # Assignment
y = 5

print(x == y)  # Comparison → True
```

So, `=` is for **storing values**, while `==` is for **checking equality**.

---

7. List and explain the standard data types available in Python.

Python provides several standard built-in data types:

1. Numeric Types

- **int** – Represents whole numbers (e.g., 10, -5)
- **float** – Represents decimal numbers (e.g., 3.14)

- **complex** – Represents complex numbers (e.g., 2 + 3j)

2. Sequence Types

- **str (String)** – Used to store text (e.g., "Hello")
- **list** – Ordered and changeable collection (e.g., [1, 2, 3])
- **tuple** – Ordered but unchangeable collection (e.g., (1, 2, 3))

3. Set Type

- **set** – Unordered collection of unique elements (e.g., {1, 2, 3})

4. Mapping Type

- **dict (Dictionary)** – Stores data in key-value pairs (e.g., {"name": "Ram"})

5. Boolean Type

- **bool** – Represents logical values: True or False

6. None Type

- **NoneType** – Represents absence of value (`None`)

---

8. Explain the difference between for loop and while loop with example programs.

A **for loop** is used when the number of iterations is known. It is commonly used to iterate over sequences like lists, strings, or ranges.

A **while loop** is used when the number of iterations is not known in advance. It continues to execute as long as the given condition is true.

Example of for loop:

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

Example of while loop:

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

Thus, a for loop is **sequence-based**, while a while loop is **condition-based**.

---

9. Write the difference between if, if–else, and if–elif–else with suitable example programs.

The **if statement** executes a block of code only when a condition is true.

The **if–else statement** provides two possible paths — one if the condition is true and another if it is false.

The **if–elif–else statement** is used when there are multiple conditions to check. Python evaluates them in order and executes the first true block.

Example:

```
marks = 75

if marks > 90:
    print("Grade A")
elif marks > 60:
    print("Grade B")
else:
    print("Grade C")
```

---

10. Define a fruitful function and explain the role of return values.

A **fruitful function** is a function that returns a value using the `return` statement. Unlike functions that only perform actions, fruitful functions send results back to the caller.

Role of return values:

- They pass computed results back to the main program.
- They allow reuse of function results.
- They help make programs modular and efficient.

Example:

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

result = add(5, 3)
print(result)
```

---

11. Define local variables and describe their scope inside a function.

A **local variable** is a variable declared inside a function. It can only be accessed within that function.

Scope of local variables:

- Exists only during the function execution.
- Cannot be accessed outside the function.
- Gets destroyed once the function finishes execution.

Example:

```
def my_function():
    x = 10  # Local variable
    print(x)

my_function()
```

Trying to use `x` outside the function will cause an error.

---

12. Define global variables and state how they are accessed.

A **global variable** is declared outside all functions and can be accessed throughout the program.

To modify a global variable inside a function, the `global` keyword must be used.

Example:

```
x = 20  # Global variable

def show():
    print(x)

show()
```

Modifying global variable:

```
x = 5
def change():
    global x
    x = 10

change()
print(x)
```

---

13. Explain in detail the purpose of built-in functions for beginners.

Built-in functions are pre-defined functions provided by Python to perform common tasks easily. They save time and reduce the need to write complex code from scratch.

Some important built-in functions include:

- `print()` – Displays output on the screen
- `input()` – Takes user input
- `len()` – Returns length of a string or list
- `type()` – Shows the data type of a variable
- `range()` – Generates a sequence of numbers
- `sum()` – Adds elements of a list
- `max()` and `min()` – Find largest and smallest values

Purpose:

- Simplify programming
- Improve code readability
- Increase development speed
- Avoid repeating common logic

---

14 Compare break and continue and describe their effect on loops.

The **break** statement immediately terminates the loop and transfers control outside the loop.

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

Example using break:

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

Example using continue:

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

---

15. Describe a practical situation where triple quotes are used in Python.

Triple quotes (`'''` or `"""`) are used for writing **multi-line strings** and **documentation strings (docstrings)**.

1. Multi-line string example:

```
message = """Welcome to Python
This is a multi-line string
used for long text."""
```

2. Docstring example:

```
def add(a, b):
    """This function returns the sum of two numbers"""
    return a + b
```

Docstrings are used to explain the purpose of functions, classes, and modules. They improve code readability and are used by documentation tools.
