PP CAT 1 Answers

1.b) Explain in brief different types of Operators in Python.

Ans.1.b)

The Different types of Operators in Python:

1. **Arithmetic operators:**
   Used for performing mathematical operations such as addition (+), subtraction (-), multiplication (*), division (/), modulus (%), exponentiation (**), and floor division (//).

2. **Assignment operators:**
   Used to assign values to variables. Examples include the assignment operator (=), as well as compound assignment operators like +=, -=, *=, /=, %=, //=, **=, &=, |=, ^=, >>=, and <<=.

3. **Comparison operators:**
   Used to compare two values. These include the equal to (==), not equal to (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=) operators.

4. **Logical operators:**
   Used to combine conditional statements. The logical operators include and, or, and not.

5. **Identity operators:**
   Used to compare objects to check if they are the same object or not. The identity operators are is and is not.

6. **Membership operators:**
   Used to test if a sequence is present in an object or not. The membership operators include in and not in.

7. **Bitwise operators:**
   Used to perform operations on binary numbers. These operators include AND (&), OR (|), XOR (^), NOT (~), left shift (<<), and right shift (>>).

1.c) Which are the build in data types in python-and how we can declare them in the program.

Ans.1.c)

In Python, there are several built-in data types that can be used to store different kinds of data. The built-in data types in Python include:

1. Text Type:
- **str: Used to store a sequence of characters, such as words or sentences.**

2. Numeric Types:
- **int: Used to store integer values, such as 1, 2, -3, etc.**
- **float: Used to store decimal or floating-point values, such as 3.14, -0.5, etc.**
- **complex: Used to store complex numbers, such as 2 + 3j, -1 + 4j, etc.**

3. Sequence Types:
- **list: Used to store a collection of items in a specific order.**
- **tuple: Used to store an immutable (unchangeable) collection of items.**
- **range: Used to store a sequence of numbers.**

4. Mapping Type:
- **dict: Used to store key-value pairs, where each value is associated with a unique key.**

5. Set Types:
- **set: Used to store an unordered collection of unique items.**
- **frozenset: Used to store an immutable collection of unique items.**

6. Boolean Type:
- **bool: Used to store either True or False.**

7. Binary Types:
- **bytes: Used to store a sequence of bytes.**
- **bytearray: Used to store a mutable sequence of bytes.**
- **memoryview: Used to store the memory view of an object.**

8. None Type:
- **NoneType: Used to represent the absence of a value or a null value.**

To declare a variable with a specific data type, you can assign a value to it using the constructor functions of the respective data types.

For example:

str	x = "Hello World"
int	x = 20
float	x = 20.5
complex	x = 1j
list	x = ["apple", "banana", "cherry"]
tuple	x = ("apple", "banana", "cherry")
range	x = range(6)
dict	x = {"name" : "John", "age" : 36}
set	x = {"apple", "banana", "cherry"}
frozenset	x = frozenset({"apple", "banana", "cherry"})
bool	x = True
bytes	x = b"Hello"
bytearray	x = bytearray(5)
memoryview	x = memoryview(bytes(5))
NoneType	x = None

Note that in Python, you don't need to explicitly declare the data type of a variable. The data type is automatically inferred based on the value assigned to it.

2.b) Differentiate between Python Module and Python Functions.

Ans.2.b)

Sr. No.	Python Modules	Python Functions
1.	A module is a file containing a set of functions and variables that can be used in other programs.	A function is a block of code that performs a specific task.
2.	Modules are used to organize code and make it reusable.	Functions are used to break down a program into smaller, manageable pieces.
3.	Modules can contain functions, variables, classes, and other objects.	Functions can have parameters (inputs) and can return a value (output).
4.	Modules are imported using the import statement.	Functions are defined using the def keyword.
5.	Modules can be created by saving code in a file with the .py extension.	Functions are called by using their name followed by parentheses.
6.	Modules can be renamed using the as keyword during import.	Functions can be called multiple times with different arguments.
7.	Built-in modules are available in Python and can be imported whenever needed.	Functions can be defined within modules or directly in the main program.

2.c) Write a Program to find area and circumference of circle by giving the radius as input from user end.

Ans.2.c)

```
import math

def calculate_circle(radius):
    area = math.pi * radius**2
    circumference = 2 * math.pi * radius
    return area, circumference

radius = float(input("Enter the radius of the circle: "))
circle_area, circle_circumference = calculate_circle(radius)

print("Area of the circle:", circle_area)
print("Circumference of the circle:", circle_circumference)
```

3.b) Differentiate between Method Overloading and Method Overriding.

Ans. 3.b)

Sr. No.	Method overloading	Method overriding
1.	Compile-time polymorphism.	Run-time polymorphism.
2.	Helps to increase the readability of the program.	Used to grant the specific implementation of the method which is already provided by its parent class or superclass.
3.	It occurs within the class.	It is performed in two classes with inheritance relationships.
4.	May or may not require inheritance.	Always needs inheritance.
5.	Methods must have the same name and different signatures.	Methods must have the same name and same signature.
6.	The return type can or cannot be the same, but we just have to change the parameter.	The return type must be the same or co-variant.
7.	Static binding is being used	Dynamic binding is being used
8.	Poor performance due to compile time polymorphism.	It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime.
9.	Private and final methods can be overloaded.	Private and final methods can’t be overridden.
10.	The argument list should be different while doing method overloading.	The argument list should be the same in method overriding.

3.c) Write a Python Program using Function to calculate the factorial of a number.

Ans.3.c)

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

num = int(input("Enter a number: "))
result = factorial(num)
print("The factorial of", num, "is", result)
```

4.b) What is mean by inheritance and explain different types of inheritance.

Ans.4.b)

Inheritance is a fundamental concept in object-oriented programming (OOP) languages like Python. It allows a class to inherit properties and methods from another class, known as the base class or parent class. The class that inherits these properties and methods is called the derived class or child class.

In Python, there are different types of inheritance:

1. **Single Inheritance:**
   In single inheritance, a derived class inherits properties and methods from a single base class. It forms a one-to-one parent-child relationship. The derived class can access all the public and protected members of the base class.

2. **Multi-Level Inheritance:**
   Multi-level inheritance occurs when a derived class inherits from another derived class. It forms a hierarchical parent-child relationship. The derived class can access the properties and methods of both the immediate parent class and the base class.

3. **Multiple Inheritance:**
   Multiple inheritance allows a derived class to inherit properties and methods from multiple base classes. It forms a multiple parent-child relationship. The derived class can access the properties and methods of all the base classes. However, conflicts may arise if multiple base classes have methods with the same name.

4. **Hierarchical Inheritance:**
   Hierarchical inheritance occurs when multiple derived classes inherit from a single base class. It forms a one-to-many parent-child relationship. Each derived class can access the properties and methods of the base class.

5. **Hybrid Inheritance:**
   Hybrid inheritance is a combination of multiple inheritance and hierarchical inheritance. It involves multiple base classes and multiple derived classes, forming a complex inheritance structure.

These different types of inheritance provide flexibility and code reusability in Python. They allow for the creation of class hierarchies and the sharing of common properties and methods among classes.

4.c) Write a Python program to create a calculator class. Include methods for basic arithmetic operations.

Ans.4.c)

```
class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

    def multiply(self, a, b):
        return a * b

    def divide(self, a, b):
        if b != 0:
            return a / b
        else:
            return "Error: Division by zero is not allowed"

# Create an instance of the Calculator class
calc = Calculator()

# Perform arithmetic operations
print("Addition:", calc.add(5, 3))
print("Subtraction:", calc.subtract(5, 3))
print("Multiplication:", calc.multiply(5, 3))
print("Division:", calc.divide(5, 3))
```

Output:

```
Addition: 8
Subtraction: 2
Multiplication: 15
Division: 1.6666666666666667
```

5.b) What are the attributes of pandas series, explain with examples.

Ans.5.b)

A Pandas Series is a one-dimensional labelled array capable of holding any data type. It's similar to a column in a spreadsheet or a single column in a Data Frame. Here are some common attributes of a Pandas Series, explained with examples:

1. **Values:**
   - **Access the underlying data as a NumPy array using the values attribute.**

```
import pandas as pd
data = [1, 2, 3, 4, 5]
series = pd.Series(data)
print(series.values)
```

2. **Index:**
   - **Access the index labels of the Series using the index attribute.**

```
import pandas as pd
data = [1, 2, 3, 4, 5]
index_labels = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index_labels)
print(series.index)
```

3. **dtype:**
   - **Retrieve the data type of the values in the Series using the dtype attribute.**

```
import pandas as pd
data = [1, 2, 3, 4, 5]
series = pd.Series(data)
print(series.dtype)
```

4. **Size:**
   - **Get the number of elements in the Series using the size attribute.**

```
import pandas as pd
data = [1, 2, 3, 4, 5]
series = pd.Series(data)
print(series.size)
```

5. **Shape:**
   - **Get the shape of the Series (number of rows) using the shape attribute.**

```
import pandas as pd
data = [1, 2, 3, 4, 5]
series = pd.Series(data)
print(series.shape)
```

These are just a few of the attributes of a Pandas Series. Depending on your use case, you may find other attributes and methods helpful for data manipulation and analysis.

6.b) Explain in detail what are the operations to be performed on arrays using numpy library.

Ans.6.b)

The NumPy library provides a wide range of operations that can be performed on arrays. Some of the common operations include:

1. **Array creation:**
   Creating arrays using the array() function or other specialized functions like zeros(), ones(), arange(), etc.

2. **Array indexing:**
   Accessing and modifying individual elements of an array using index numbers.

3. **Array slicing:**
   Extracting a portion of an array by specifying start and end indices.

4. **Array reshaping:**
   Changing the shape or dimensions of an array using the reshape() function.

5. **Array concatenation:**
   Combining multiple arrays into a single array using functions like concatenate(), stack(), hstack(), vstack(), etc.

6. **Array arithmetic:**
   Performing mathematical operations on arrays such as addition, subtraction, multiplication, division, etc.

7. **Array broadcasting:**
   Performing operations between arrays of different shapes by automatically aligning their dimensions.

8. **Array aggregation:**
   Computing statistical measures like mean, median, sum, min, max, etc. on arrays or specific axes of arrays.

9. **Array sorting:**
   Sorting the elements of an array using functions like sort() or argsort().

10. **Array manipulation:**
    Manipulating arrays by adding or removing elements, changing data types, transposing, etc.

These are just a few examples of the operations that can be performed on arrays using the NumPy library. NumPy provides a comprehensive set of functions and methods for efficient array manipulation and computation.