Python Advanced Assignment:-

Solution of Assignment-1 :-

Submitted By:- Ambarish Singh

Q.1) What is the purpose of Python's OOP ?

Ans.1)

The purpose of Python's Object-Oriented Programming (OOP) is to provide a way to organize and structure code by modeling real-world objects and their relationships. OOP allows for the creation of objects, which are instances of classes, and classes provide a blueprint for creating objects with specific properties and behaviors.

Some of the benefits of using OOP in Python include:

a) **Encapsulation:** The ability to hide the internal workings of objects, so that they can be used without revealing their implementation details.

b) **Abstraction:** The ability to define interfaces and abstract classes that provide a high-level view of the functionality of a class, without specifying the implementation details.

c) **Inheritance:** The ability to create subclasses that inherit properties and behaviors from a parent class, and to customize or extend the functionality of those classes.

d) **Polymorphism:** The ability to use objects of different classes in a similar way, by defining common interfaces or by using inheritance.

Using OOP in Python can make code more modular, reusable, and maintainable, and can help developers to write more efficient and effective code.

Q.2) Where does an inheritance search look for an attribute?

Ans.2)

In Python, when an object tries to access an attribute (e.g., a method or a variable), Python's inheritance search looks for that attribute in the following order:

a) **The object itself:** If the attribute is present in the object, the search stops and the attribute is used.

b) **The class of the object:** If the attribute is not present in the object, Python looks for the attribute in the class of the object.

c) **The superclasses of the class:** If the attribute is not present in the class of the object, Python looks for the attribute in the superclasses of the class, in the order they were specified in the class definition.

d) **The module of the class:** If the attribute is not present in any of the superclasses, Python looks for the attribute in the module of the class.

e) **The built-in object:** If the attribute is not present in the module of the class, Python looks for the attribute in the built-in object.

If the attribute is not found in any of these locations, Python raises an AttributeError.

This search order is often referred to as the Method Resolution Order (MRO). The MRO can be influenced by method overriding, super calls, and the order in which multiple inheritance is specified.

Q.3) How do you distinguish between a class object and an instance object?

Ans.3)

In Python, a class is a blueprint or a template for creating objects, while an instance is an individual object created from a class.

To distinguish between a class object and an instance object, consider the following:

- A class object is created when a class is defined in the code. It is used to create instances of the class and can be used to access class-level properties and methods.

- An instance object is created when a class is instantiated. It is a unique object that has its own set of instance-level properties and can call instance-level methods.

- The class object defines the attributes and behaviors that the instance object will have, but the instance object is a specific, unique object that can have its own state.

In summary, a class object is a general representation of a class that can be used to create instance objects, while an instance object is a specific, unique object that is created from a class and has its own set of attributes and behaviors.

Q.4) What makes the first argument in a class's method function special ?

Ans.4)

In Python, the first argument of a class method function is usually referred to as "self". This is a convention that indicates the instance of the class that the method is being called on. The self argument is automatically passed to the method when it is called, and it is used to access the attributes and methods of the instance.

The self argument is what makes instance methods different from class methods and static methods. Instance methods are defined with a self parameter, which represents the instance on which the method is being called. This allows the method to access and modify the state of the instance.

For example, consider the following class definition:

Python Code :-

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"My name is {self.name} and I am {self.age} years old.")
```

In this example, the "self" parameter in the "init" and "introduce" methods refers to the instance of the "Person" class on which the method is being called. When an instance of the "Person" class is created and the "introduce" method is called on it, the "self" parameter is automatically

passed to the method, and it allows the method to access the "name" and "age" attributes of the instance.

In summary, the "self" parameter in a class method function is what allows the method to access and modify the attributes of the instance on which it is called.

Q5. What is the purpose of the __init__ method ?

Ans.5)

The "init" method in Python is a special method that is called when an object is created from a class. Its purpose is to initialize the object's attributes and set them to their initial values.

The "init" method is also referred to as the constructor method, as it is used to construct an object from the class. It takes in the "self" parameter as its first argument, which refers to the instance of the object being created. Other parameters can also be added to the method signature to set initial values for the object's attributes.

For example, consider the following class definition:

Python Code:-

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

In this example, the "init" method takes in two parameters, "name" and "age", and sets the corresponding attributes of the instance to their initial values.

When an instance of the "Person" class is created, the "init" method is automatically called with the instance as the "self" argument, and the "name" and "age" arguments are used to set the initial values of the instance's attributes.

Using the "init" method allows us to create instances with predefined attributes, and ensures that the object is in a valid state from the moment it is created. This makes it easier to work with the object and helps avoid errors that may arise from uninitialized attributes.

Q.6) What is the process for creating a class instance?

Ans.6)

To create an instance of a class in Python, you follow these steps:

a) **Define the class:** First, you need to define the class by specifying the attributes and methods that it will have.

b) **Instantiate the class:** To create an instance of the class, you need to call the class and assign the result to a variable. The call to the class is made like a function call, passing any required arguments to the class constructor method (init).

For example, consider the following class definition:

Python Code :-

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

To create an instance of the "Person" class, you can call it with the required arguments, like this:

Python Code :-

```
person1 = Person("Alice", 30)
```

This creates an instance of the "Person" class and assigns it to the "person1" variable. The "init" method is automatically called with the instance as the "self" argument, and the "name" and "age" arguments are used to set the initial values of the instance's attributes.

You can create multiple instances of the same class, each with its own set of attributes and methods, by repeating this process.

c) **Access the instance's attributes and methods:** Once you have created an instance of a class, you can access its attributes and methods using the dot notation. For example, to access the "name" attribute of the "person1" instance, you can use the following code:

Python Code :-

```
print(person1.name)
```

This will print "Alice", which is the value of the "name" attribute of the "person1" instance.

In summary, creating an instance of a class involves defining the class, instantiating the class by calling it with any required arguments, and then accessing the instance's attributes and methods using the dot notation.

Q.7) What is the process for creating a class?

Ans.7)

To create a class in Python, you can follow these steps:

i) Define the class: First, you need to define the class by giving it a name and specifying the attributes and methods that it will have. This is done using the "class" keyword, followed by the class name and a colon (:). For example, here's a simple class definition for a "Person":

Python Code:-

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"My name is {self.name} and I am {self.age} years old.")
```

ii) Define the attributes and methods: After defining the class, you can define the attributes and methods that the class will have. In the example above, the "Person" class has two attributes, "name" and "age", and one method, "introduce()".

iii) Instantiate the class: Once the class has been defined, you can create instances of the class by calling it and assigning the result to a variable. For example, to create an instance of the "Person" class, you can use the following code:

Python Code :-

```
person1 = Person("Alice", 30)
```

This creates an instance of the "Person" class and assigns it to the "person1" variable. The "init" method is automatically called with the instance as the "self" argument, and the "name" and "age" arguments are used to set the initial values of the instance's attributes.

iv) Access the instance's attributes and methods: Once you have created an instance of a class, you can access its attributes and methods using the dot notation. For example, to access the "name" attribute of the "person1" instance, you can use the following code:

Python Code:-

```
print(person1.name)
```

This will print "Alice", which is the value of the "name" attribute of the "person1" instance. To call the "introduce()" method of the "person1" instance, you can use the following code:

Python Code:-

```
person1.introduce()
```

This will print "My name is Alice and I am 30 years old.", which is the output of the "introduce()" method of the "person1" instance.

In summary, creating a class involves defining the class by giving it a name and specifying its attributes and methods, instantiating the class by calling it and assigning the result to a variable, and then accessing the instance's attributes and methods using the dot notation.

Q.8) How would you define the superclasses of a class?

Ans.8)

In object-oriented programming, a superclass (or parent class) is a class that is being extended or subclassed by another class. The superclass provides a blueprint for the subclasses to inherit methods, attributes, and behavior. The subclasses can also override or add new methods and attributes to the superclass.

In simpler terms, a superclass is a more general or abstract class that defines common properties and behaviors that can be shared by its subclasses. The subclasses then inherit these properties and behaviors and can also add their own unique properties and behaviors.

For example, in a program that models different types of vehicles, a superclass could be "Vehicle," and subclasses could be "Car," "Truck," and "Motorcycle." The "Vehicle" class could define common properties such as "wheels," "fuel type," and "maximum speed," which the subclasses would inherit. The subclasses could then define their own unique properties such as "number of doors" for "Car" and "payload capacity" for "Truck."

========== Thank You ==========