Q1. What is the purpose of Python's OOP?

Answer:

OOP stands for Object-Oriented Programming. It is a style of programming characterized by the identification of classes of objects closely linked with the methods (functions) with which they are associated. The purpose of Python’s OOP describes in the following:

* Object-oriented Programming (OOPs) uses objects and classes in programming
* It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming
* The principle of OOP is to bind the data and the functions to work together as a single unit so that no other part of the code can access this data
* It helps to divide the overall program into different small segments and thus making it solving easy with the use of objects
* It helps in easy maintenance and modification of existing program

Q2. Where does an inheritance search look for an attribute?

**Answer:**

An inheritance search looks for an attribute first in the instance object, then in the class the instance was created from, then in all higher superclasses, progressing from left to right.

Q3. How do you distinguish between a class object and an instance object?

Answer:

|  |  |
| --- | --- |
| Class object | Instance object |
| Class is used as a template for declaring and creating the objects. | An object is an instance of a class. |
| Memory is not allocated when a class is created. | Objects are allocated memory space whenever they are created. |
| The class can be declared only once. | An object is created many times as per requirement. |
| A class cannot be manipulated as they do not have allocated memory. | Objects can be manipulated. |
| A class is a logical entity. | An object is a physical entity. |
| Class does not contain any values which can be associated with the field. | Each object has its own values, which are associated with it. |
| A class is used to bind data as well as methods together as a single unit. | Objects are like a variable of the class. |

Q4. What makes the first argument in a class’s method function special?

Answer:

The first argument or parameter in a class’s method function is self. The self is used to represent an instance (object) of the given class. It is always pointing to current object. We can access the attributes and methods of the class in python. It also binds the attributes with the given arguments. It must be provided as a first parameter to the instance method and constructor otherwise it will cause an error.

Q5. What is the purpose of the \_\_init\_\_ method?

Answer:

The \_\_init\_\_ method is declared within a class and is used to initialize (assign values) the attributes of an object as soon as the object is formed. It is a constructor contains collection of statements that are executed at the time of object creation. This method is called when an object is created from a class and it allows the class to initialize the attributes of the class.

Q6. What is the process for creating a class instance?

Answer:

Process of creating a class instance:

--> call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts.

--> example:

class\_1 **=** test\_class("A", "B")

Here, class\_1 is an instance of class test\_class with attributes "A" and "B".

Q7. What is the process for creating a class?

Answer:

A keyword called class is using to defining a new class. The class name is declared immediately after the class keyword followed by a colon. The syntax of the class definition is as follow:

class <class name>:

Example of creating a class:

|  |
| --- |
| class Student:  def \_\_init\_\_(self, name):  self.name = name  def display\_name(self):  return self.name  def \_\_str\_\_(self):  return f"This is a student class"    s1 = Student("Debabrata")  print(s1.display\_name()) |

Q8. How would you define the superclasses of a class?

Answer:

A superclass/parent class is the class from which many subclasses/child classes can be created. The subclasses inherit the characteristics of a superclass.

Example:

|  |
| --- |
| class Parent:  def method\_1(self):  return "this is method of class parent"    class Child\_1(Parent):  def method\_2(self):  return "this is method of class child-1"  class Child\_2(Parent):  def method\_3(self):  return "this is method of class child-2"    # Class Child\_1, Child\_2 are subclasses and Class Parent is superclass  a = Parent()  b = Child\_1()  c = Child\_2()  print(a.method\_1())  print(b.method\_2())  print(b.method\_1())  print(c.method\_3())  print(c.method\_1())  # Objects of class Child\_1 and Child\_2 are able to access method of class Parent because Class Parent is superclass. |