Q1. What is the relationship between classes and modules?

Answer:

|  |  |
| --- | --- |
| Classes | Modules |
| Classes in python act as a blueprint based on which objects are created. | Modules in Python are files with a .py extension using which we can reuse elements inside that file. |
| Classes can be defined in our main application code or inside modules imported by our application. | A python module is nothing but a package to encapsulate reusable code. Modules usually, but not always, reside in a folder with a \_\_init\_\_.py file inside of it. |
| Classes are the code of Object Oriented Programming and can contain properties and methods. | Modules can contain functions but also classes. Modules are imported using the import keyword. |
| Classes can hold its state (by keeping track of instance variables) and can be duplicate as many times as we want. | We do not instantiate modules, instead we just use methods in them. |

Q2. How do you make instances and classes?

Answer:

To create an instances we call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts and to create a class we use class definition where we write class keyword followed by class name and then colon.

Example:

|  |
| --- |
| class Employees:  # using the constructor  def \_\_init\_\_(self, employee\_name, employee\_salary):  # using instance variable  self.name = employee\_name  self.salary = employee\_salary    # accessing the instance variable from instance method  def display(self):  print(f'Employee Name: {self.name}, Employee Salary: {self.salary}')  # here, we are creating the first object  emp1 = Employees("X", 45000)  # calling the instance method  emp1.display()  # here, we are creating the second object  emp2 = Employees("Y", 55000)  # calling the instance method  emp2.display() |

Q3. Where and how should be class attributes created?

Answer:

Class attributes belong to the class itself they will be shared by all the instances. These attributes are the variables defined directly in the class that are shared by all objects of the class. Such attributes are defined in the class body parts usually at the top, for legibility.

Example:

|  |
| --- |
| class countclass:  count = 0 # class attribute    def incresecount(self):  countclass.count += 1    # Calling incresecount method  s1 = countclass()  s1. incresecount()  print(s1.count)    # Calling incresecount on one more object  s2 = countclass ()  s2. incresecount ()  print(s2.count)    print(countclass.count) |

Q4. Where and how are instance attributes created?

Answer:

Instance attributes are defined in the constructor. Defined directly inside a class. Defined inside a constructor using the self parameter. Shared across all objects. This attribute is a Python variable belonging to only one object. Unlike class attributes, instance attributes are not shared by objects. Every object has its own copy of the instance attribute. They are created in \_\_init\_\_ method of the class.

Example:

|  |
| --- |
| # Setting Attribute Values  class Student:  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age    # Passing Instance Attribute Values in Constructor  std = Student('Debabrata',35)  # attributes are accessed by objects  print(std.name)  print(std.age)  std.name = 'Ram'  std.age = 45  # attributes are accessed by objects  print(std.name)  print(std.age) |

Q5. What does the term "self" in a Python class mean?

Answer:

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. This keyword allows us to access variables, attributes, and methods of a defined class in Python.

|  |
| --- |
| class Car:  def \_\_init\_\_(self, color, price, engine):  # these are instance attributes  self.color = color  self.price = price  self.engine = engine  car1 = Car('Honda City', ‘13.42 lakhs’, '1498cc')  car2 = Car('BMW 7 Series', ‘1.96 crore’, '2998cc')  print(car1.\_\_dict\_\_)  print(car2.\_\_dict\_\_) |

Q6. How does a Python class handle operator overloading?

Answer:

The operator overloading in Python generally provide extended meaning beyond their predefined operational meaning. As for example, we use the "+" operator for adding two integers as well as joining two strings or merging two lists. This is achieved as the "+" operator is overloaded by the "int" class and "str" class. The Python class handle operator overloading by using special methods called Magic methods, also called dunders. These special methods usually begin and ending with double underscore (\_\_).

Example of magic methods for basic arithmetic operators:

Addition: \_\_add\_\_()

Subtraction: \_\_sub\_\_()

Multiplication: \_\_mul\_\_()

|  |
| --- |
| class Point:  def \_\_init\_\_(self, x=0, y=0):  self.x = x  self.y = y  def \_\_str\_\_(self):  return "({0},{1})".format(self.x, self.y)  def \_\_add\_\_(self, other):  x = self.x + other.x  y = self.y + other.y  return Point(x, y)  p1 = Point(1, 2)  p2 = Point(2, 3)  print(p1+p2) |

Q7. When do you consider allowing operator overloading of your classes?

Answer:

When there is a situation that we have to use same opereator to represent different meaning we need to use operator overloading.

Q8. What is the most popular form of operator overloading?

Answer:

The most popular form of operator overloading in Python is special methods which are also called Python magic methods or dunders that are denoted or define by double underscore in both side of the method name.

Q9. What are the two most important concepts to grasp in order to comprehend Python OOP code?

Answer:

The two most important and main aspect of OPPs are class and object. Along with this the four principles of OOPs are Inheritance, Polymorphism, Encapsulation, and Abstraction.