**Q1. What is the relationship between classes and modules?**

Both are the combination of modules and constants. A class can generate instances of objects and instance variables. Module cannot generate instances. Generally a module is called through import statement, wheras class is called by its class name and instance name.

**Q2. How do you make instances and classes?**

To create instances of a class, you call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts

class Employee:

'Common base class for all employees'

empCount = 0

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

emp1 = Employee("Zara", 2000)

emp2 = Employee("Manni", 5000)

**Q3. Where and how should be class attributes created?**

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

Class sample:

Cl1= 54

Cl2=106

**Q4. Where and how are instance attributes created?**

Unlike class attributes, instance attributes are not shared by objects. Every object has its own copy of the instance attribute (In case of class attributes all object refer to single copy).

class sample:

  def \_\_init\_\_(self,name):

    self.name=name

sample1=sample("xyz")

print(sample1.name)

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

It is the first argument in class method function. The first argument allows the access to the attributes and methods of each object. Multiple number of objects are given access through this argument (self). If there is no self argument the same class could not hold information of one or more objects that are created.

**Q6. How does a Python class handle operator overloading?**

The operator overloading in Python means provide extended meaning beyond their predefined operational meaning. Such as, we use the "+" operator for adding two integers as well as joining two strings or merging two lists. We can achieve this as the "+" operator is overloaded by the "int" class and "str" class. The user can notice that the same inbuilt operator or function is showing different behavior for objects of different classes. This is called operator overloading. To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined.

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

When we redefine the behavior of the existing operator, we allow operator overloading by defining a special function.

class A:

    def \_\_init\_\_(self, a):

        self.a = a

    # adding two objects

    def \_\_add\_\_(self, o):

        return self.a + o.a

ob1 = A(1)

ob2 = A(2)

ob3 = A("Geeks")

ob4 = A("For")

print(ob1 + ob2)

print(ob3 + ob4)

# Actual working when Binary Operator is used.

print(A.\_\_add\_\_(ob1 , ob2))

print(A.\_\_add\_\_(ob3,ob4))

#And can also be Understand as :

print(ob1.\_\_add\_\_(ob2))

print(ob3.\_\_add\_\_(ob4))

Here, we defined the special function “\_\_add\_\_( )”  and when the objects ob1 and ob2 are coded as “ob1 + ob2“, the special function is automatically called as **ob1.\_\_add\_\_(ob2)**which simply means that ob1 calls the \_\_add\_\_( ) function with ob2 as an Argument and It actually means **A .\_\_add\_\_(ob1, ob2)**. Hence, when the Binary operator is overloaded, the object before the operator calls the respective function with object after operator as parameter.

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

Binary overloading is the most popular form of operator overloading

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

1. Inheritence
2. Polymorphism