Q1. What is the relationship between classes and modules?

No particular relation between classes and modules in theory.

Now, a class is simply a special way of writing a object constructors. Basically it’s a function, just written in a certain way. So when exporting a “class”, you’re exporting a function. A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

When modules are used ,we “import” a module, then use it. We can import only certain “parts” (exports) of a module, you can import the default export, if the module has one, but nothing more to it. When writing a module, we export things(like - a string, a number, a null, a function, class, object) out of it. We can even make one of the things you exported the default export, but not necessarily.

Now, in practice, in some environments (Angular 2 comes to mind), your modules will typically have only one export, and that export will be a class. So in those cases, there’s almost a one-to-one relation with modules and classes, but I don’t think it ever is fully one-to-one.

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. You access the object's attributes using the dot operator with object. Class variable would be accessed using class name.

Class Employee():

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

def display(self):

print(self.name)

print(self.idnumber)

emp1 = Employee("Zara", 2000)

emp1.displayEmployee()

Q3. Where and how should be class attributes created?

**Class attributes** are the variables defined directly in the class that are shared by all objects of the class.

To define a class attribute, you place it**outside of the \_\_init\_\_ () method**. Use class\_name.class\_attribute or object\_name.class\_attribute to access the value of the class\_attribute

class Cat:

# class attribute

attr = "mammal"

# Instance attribute

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

self.name = name

Q4. Where and how are instance attributes created?

**Instance attributes** are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.

# Instance attribute

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

self.name = name

| Class Attribute | Instance Attribute |
| --- | --- |
| Defined directly inside a class. | Defined inside a constructor using the self parameter. |
| Shared across all objects. | Specific to object. |
| Accessed using class name as well as using object with dot notation, e.g. classname.class\_attribute or object.class\_attribute | Accessed using object dot notation e.g. object.instance\_attribute |
| Changing value by using classname.class\_attribute = value will be reflected to all the objects. | Changing value of instance attribute will not be reflected to other objects. |

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

self represents the instance of the class. By using the “self”  we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.Python does not use the @ syntax to refer to instance attributes. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on.

Q6. How does a Python class handle operator overloading?

**Operator Overloading** means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class. the same built-in operator or function shows different behavior for objects of different classes, this is called *Operator Overloading*.

Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary ‘+’ operator it throws an error, because compiler don’t know how to add two objects. So we define a method for an operator and that process is called 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.

# Python Program illustrate how

# to overload an binary + operator

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)

output :

3

GeeksFor

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

One can overload the + operator to concatenate your user-defined string class, or add two Fraction class objects together or overload the << operator to make it easy to print your class to the screen (or a file) or overload the equality operator (==) to compare two class objects.

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

The most frequent instance is the**adding up operator ‘+’**, where it can be used for the usual addition and also for combining two different strings.

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

**inheritance and polymorphism**