Q1. What is the relationship between classes and modules?

Classes are simply the blueprint of an object. Modules are just python files with .py extension. Modules can have functions, variables, and even classes. Modules can be imported on any python program. We can instantiate an object in the program in which we imported the module having a class.

Q2. How do you make instances and classes ?

Instances are the objects we instantiate. These instances are examples of a class. Classes can be created using the class keyword.

Example:

Class Employee:

def init(self, name):

pass

Obj = Employee(‘john doe)

Q3. Where and how should class attributes be created?

| class Employee:  company = 'ineuron'  def init(self):  pass  obj.Employee()  Employee.company  obj.company |
| --- |

Here company is the class attribute. We can access it using class name and object.

It is created within a class.

Q4. Where and how are instance attributes created?

Instance attributes are defined in \_\_init\_\_() of class.

| class Student:  def \_\_init\_\_(name, age):  self.name = name  self.age = age |
| --- |

We pass the values to the instance attribute while object instantiation.

| std1 = Student('samplename', 20) |
| --- |

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

Self keyword refers to the current object. It stores the memory address of the current object. It acts as a pointer to that object.

Q6. How does a Python class handle operator overloading?

We can overload the operators using some special functions. To perform addition of two variables, we can use a function called \_\_add\_\_().

a.\_\_add\_\_(b)

Here we overload + operator with function \_\_add\_\_()

Similarly to perform subtraction, we use \_\_sub\_\_() function,

a.\_\_sub\_\_(b)

Here we overload - operator with function \_\_sub\_()

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

We consider the same when we have to perform arithmetic operations of 2 objects like addition, subtraction,...

| class Student:  def \_\_init\_\_(self, mark1, mark2):  self.mark1 = mark1  self.mark2 = mark2  obj1 = Student(20, 21) obj2 = Student(23, 24) print(obj1 + obj2) |
| --- |

It raises an error like,

TypeError: unsupported operand type(s) for +: 'Student' and 'Student'

We can make it work with the help of operator overloading.

class Student:

def \_\_init\_\_(self, mark1, mark2):

self.mark1 = mark1

self.mark2 = mark2

def \_\_str\_\_(self):

return "({0},{1})".format(self.mark1, self.mark2)

def \_\_add\_\_(self, other):

x = self.mark1 + other.mark1

y = self.mark2 + other.mark2

return Student(x, y)

obj1 = Student(20, 21)

obj2 = Student(23, 24)

print(obj1 + obj2)

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

Overloading addition operator.

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

1. Classes and Objects
2. Inheritance