Q1. What is the relationship between classes and modules?

A Python class is like an outline/blueprint/mold for creating a new object. An object is anything that you wish to manipulate or change while working through the code. Every time a class object is instantiated, which is when we declare a variable, a new object is initiated from scratch

Whereas in Python, Modules are simply files with the . py extension containing Python code that can be imported inside another Python Program. In simple terms, we can consider a module to be the same as a code library or a file that contains a set of functions/Classes that you want to include in your application.

Classes in python are templates for creating objects. They contain variables and functions which define the class objects. At the same time, modules are python programs that can be imported into another python program. Importing a module enables the usage of the module’s functions and variables into another program

Q2. How do you make instances and classes?

For creating a class instance we call a class by its name and pass the arguments whch its **\_\_init\_\_** method accepts.

**Example:** FSDS= ineuron(‘studensts,1000), FSDS is an instance of class ineuron with attriubutes 'candidates' and 10000.

Whereas for creating a class, we use the **Class** keyword. Class keyword is followed by classname..

**Example:** Here ineuron is a class created with class keyword with students and count.

class Employee:

def \_\_init\_\_(self, students,1000):

self.students= students

self.count = count

Q3. Where and how should be class attributes created?

Class attributes are attributes which are owned by the class itself. They will be shared by all the instances of the class. Therefore they have the same value for every instance. We define class attributes outside all the methods, usually they are placed at the top, right below the class header

class Car: no\_of\_wheels = 4; # this is a class attribute

def \_\_init\_\_(self,color,price,engine):

self.color = color # All this are instance attributes

self.price = price self.engine = engine

Q4. Where and how are instance attributes created?

Instances attributes are passed to the class when an object of the class is created. Unlike class attributes, instance attributes are not shared by all objects of the classs. instead each object maintains its own copy of instance attributes at object level. whereas incase of class attributes all instances of class refer to a single copy. Usually instance attributes are defined within the **\_\_init\_\_** method of class

**Example:** In the below sample code we are creating a class Car with instance varaibles color, price, engine, which will be provided when an instance of class Car is created.

class Car:

def \_\_init\_\_(self,color,price,engine):

self.color = color # All this are instance attributes

self.price = price

self.engine = engine

TATA = Car('Gray', 700000, 'electric')  
Toyota = Car('Blue',1800000, 'petrol')

TATA& Toyota are both the instances of class Car with different instance variables.

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.

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

Q6. How does a Python class handle operator overloading?

Python Classes handle operator overloading by using special methods called **Magic methods**. these special methods usually begin and end with **\_\_**

**class** dairy:

**def** \_\_init\_\_(self,pages):

self**.**pages **=** pages

**def** \_\_add\_\_(self,preface):

**return** self**.**pages **+** preface**.**pages

p1 **=** dairy(500)

p2 **=** dairy(600)

print(f'The total number of pages in dairies is {p1**+**p2}')

The total number of pages in 2 books is 1100

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

 When we want to have different meaning for the same operator accroding to the context we use operator overloading.

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

The most popular form of operator overloading in python is by special methods called **Magic methods**. Which usually beign and end with double underscore **\_\_<method name>\_\_**.

class Sum:

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

self.k = k

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

return self.k+n.k

obj1 = Sum(1)

obj2 = Sum(2)

print(f'Sum -> {obj1+obj2}')

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

**Classes** and **objects** are the two concepts to comprehend python OOP code as more formally objects are entities that represent instances of general abstract concept called class

Important concepts are,

Inheritance

Abstraction

Polymorphism

Encapsulation