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.

Q2. How do you make instances and classes?

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

**Example:** vishwak = employee('Male',20000), Here vishwak is an instance of class employee with attriubutes 'Male' and 20000.

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

**Example:** Here Employee is a class created with class keyword with arguments gender and salary.

class Employee: def \_\_init\_\_(self, gender,salary): self.gender = gender self.salary = salary

Q3. Where and how should be class attributes created?

Class attributes or Class level Attributes belong to the class itself. these attributes will be shared by all the instances of the class. Hence these attributes are usually created/defined in the top of class definiation outside all methods.

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.

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

**SELF** is a default variable that contains the memory address of the current object. Instance variables and methods can be referred to by the self variable. When the object of a class is created, the memory location of the object is contained by its object name. This memory location is passed to the SELF internally, as SELF knows the memory address of the object, so the variable and method of an object is accessible. The first argument to any object method is SELF because the first argument is always object reference. This process takes place automatically whether you call it or not.

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 **\_\_** (double underscore)  
**Example:** Magic methods for basic arithmetic operators are:

* + -> \_\_add\_\_()
* - -> \_\_sub\_\_()
* \* -> \_\_mul\_\_()
* / -> \_\_div\_\_()

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>\_\_**.

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.

Along with that Inheritence, Abstraction, Polymorphism, Encapsulation are also the important concepts of python oop’s.