Q1. What is the relationship between classes and modules?

Ans:- **Modules are collections of methods and constants**. They cannot generate instances. Classes may generate instances (objects), and have per-instance state (instance variables).

**a class is used to define a blueprint for a given object**, whereas a module is used to reuse a given piece of code inside another program.

Q2. How do you make instances and classes?

Ans:-

Use the class name to create a new instance  
  
**Call ClassName()** to create a new instance of the class ClassName . To pass parameters to the class instance, the class must have an \_\_init\_\_() method. Pass the parameters in the constructor of the class.

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

Unlike **class** attributes, **instance** attributes are not shared by objects. Every object has its own copy of the **instance** attribute

Instance attributes are **defined in the constructor**. Defined directly inside a class. Defined inside a constructor using the self parameter. Shared across all objects.

Q3. Where and how should be class attributes created?

Ans:- 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 attributes are **variables of a class that are shared between all of its instances**. They differ from instance attributes in that instance attributes are owned by one specific instance of the class only, and ​are not shared between instances.

Q4. Where and how are instance attributes created?

Ans:- Instance attributes are **defined in the constructor**. Defined directly inside a class. Defined inside a constructor using the self parameter. Shared across all objects.

An instance attribute is a Python variable belonging to one, and only one, object. This variable is only accessible in the scope of this object and it is defined **inside the constructor function, \_\_init\_\_(self,..) of the class**

1. class ObjectClass():
2. def \_\_init\_\_(self):
3. self. attribute1 = "attribute1"
4. def newAttr(self, attr):
5. setattr(self, attr, attr)
6. objectClass = ObjectClass()
7. print(objectClass. attribute1)
8. setattr(objectClass, "newAttribute", "new attr")

An instance attribute is **a Python variable belonging to only one object**. It is only accessible in the scope of the object and it is defined inside the constructor function of a class.

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

Ans:- self **represents the instance of the class**. By using the “self” keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. The reason you need to use self. is because 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.

self : **an argument expected to be the instance from which the method was called**. **other** : an argument expected to be an instance of the class, but not the one calling the method.

Q6. How does a Python class handle operator overloading?

Ans:- Operator overloading in Python is **the ability of a single operator to perform more than one operation based on the class (type) of operands**. For example, the + operator can be used to add two numbers, concatenate two strings or merge two lists.

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?

Ans:- operators can be overloaded globally or **on a class-by-class basis**. Overloaded operators are implemented as functions and can be member functions or global functions. An overloaded operator is called an operator function. You declare an operator function with the keyword operator preceding the operator.

It allows you to provide an intuitive interface to users of your class, plus makes it possible for templates to work equally well with classes and built-in/intrinsic types.

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

Ans:- A very popular and convenient example is **the Addition (+) operator**. Just think how the '+' operator operates on two numbers and the same operator operates on two strings. It performs “Addition” on numbers whereas it performs “Concatenation” on strings.

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

Ans:- **There are three major features in object-oriented programming that makes them different than non-OOP languages: encapsulation, inheritance and polymorphism.**

* Encapsulation Enforces Modularity. ...
* Inheritance Passes "Knowledge" Down. ...
* Polymorphism Takes any Shape. ...
* OOP Languages.