Q1. What is the relationship between classes and modules?

**Answer :a class can be instantiated but a module cannot. A module will never be anything other than a library of methods. A class can be so much more -- it can hold its state (by keeping track of instance variables) and be duplicated as many times as you want. It's all about *objects*.**

Q2. How do you make instances and classes?

**Answer :**

**Class Variables — Declared inside the class definition (but outside any of the instance methods). They are not tied to any particular object of the class, hence shared across all the objects of the class. Modifying a class variable affects all objects instance at the same time.**

**Instance Variable — Declared inside the constructor method of class (the \_\_init\_\_ method). They are tied to the particular object instance of the class, hence the contents of an instance variable are completely independent from one object instance to the other.**

**Eg >**

**Class Car :**

**Wheels = 4 # <- Class variable**

**Def \_\_init\_\_(self,name):**

**Self.name =name # <- Instance variable**

Q3. Where and how should be class attributes created?

**Answer : class Car :**

**Wheels = 4 # <- class attributes**

Q4. Where and how are instance attributes created?

Answer :

**Class Car :**

**Def \_\_init\_\_(self,name):**

**Self.name =name # <- Instance variable**

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

**Answer :**

**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**

**Self is always pointing to current object.**

Q6. How does a Python class handle operator overloading?

**Answer :**

**In Python we have predefined function for operator like for “+” \_\_add\_\_(self,other) , when we call use this operator then python will call it’s associate function and based on class it perform operation.**

**Eg > 1 + 2 # <- here based on int class addition take place**

**‘A’ + ‘B’ # <- here based on str class concatenation take place.**

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

**Answer : Operator Overloading means giving extended meaning beyond their predefined operational meaning.**

**If we want to do addition on class object then if we use class1() + class2() then this will produce TypeError() so we can overload \_\_add\_\_(self,other) function.**

**Eg .**

**class bubble:**

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

**self.volume = volume**

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

**return "volume is " + str(self.volume)**

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

**volume = self.volume + other.volume**

**return bubble(volume)**

**b1 = Bubble(10)**

**B2 = Bubble(20)**

**b3 = b1 + b2 #<-volume is 30**

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

**Answer : 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?

**Answer : Inheritance, Encapsulation, Polymorphism, and Data abstraction**