Q1. What is the relationship between classes and modules?

Ans: The relationship between classes and modules is that classes can be defined within a module, just like functions and variables. When a class is defined in a module, it becomes part of that module's namespace and can be accessed using dot notation when the module is imported into another program.

Q2. How do you make instances and classes?

Ans:

1. Creating a class:

To create a class, use the class keyword followed by the name of the class. Here's an example:

Ex:

class class\_name:

pass

2. Creating an instance:

To create an instance of a class, you need to call the class name followed by parentheses. Here's an example:

Ex:

object = class\_name() # here if constructor is created for class then class attributes will be initialized with values.

Q3. Where and how should be class attributes created?

Ans: In Python, class attributes are variables that are shared among all instances of a class. Class attributes are typically created inside the class definition, but outside of any method definitions.

1. During object creation inside the constructor.

2. Inside class also attributes are created, like below

Ex:

class Car:

manufacturer = "Toyota"

num\_wheels = 4

def \_\_init\_\_(self, model, year):

self.model = model

self.year = year

Q4. Where and how are instance attributes created?

Ans:

1. In Python, instance attributes are variables that are specific to each instance of a class. Instance attributes are typically created inside the \_\_init\_\_ method of a class, using the self parameter.

Ex:

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

2. After object creation, outside of the class, attributes can be reinitialized using DOT notation.

Ex:

car1 = Car("Toyota", "Corolla", 2021)

print(car1.make) # using DOT notation

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

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

Ex:

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make # using self-keyword will initiate the attributes of the class

self.model = model

self.year = year

def drive(self):

print(f"Driving the {self.make} {self.model}") #using self-keyword to access the attributes make and model

Q6. How does a Python class handle operator overloading?

Ans: Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

Ex:

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

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

return Point(self.x + other.x, self.y + other.y)

p1 = Point(1, 2)

p2 = Point(3, 4)

p3 = p1 + p2

print(p3.x, p3.y) # Output: 4 6

Here, p1 and p2 objects are initialized with their respective values. When we add the p1 and p2 objects using ‘+’ operator, python will automatically calls the method \_\_add\_\_ to add the values of p1 and p2 then new Point() object is initialized with the added values of p1 and p2 then returns the object and assigned to p3 object. Later, we can access the instances of p3.

Not only, for addition of integers we can also use method overloading for strings in python.

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

Ans:

Operator overloading can be useful when it makes the code more readable and easier to understand. For example, if your class represents a mathematical concept, such as a vector or a complex number, overloading the arithmetic operators can make the code more intuitive and natural to read.

However, operator overloading can also make the code harder to read and understand if it is used inappropriately or excessively. For example, overloading operators that have non-standard meanings in your class can lead to confusion and errors.

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

Ans:

In Python, the most popular form of operator overloading is probably the arithmetic operators, such as +, -, \*, and /. Many mathematical objects can be represented as Python objects, and overloading the arithmetic operators makes it more natural and intuitive to work with these objects.

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

Ans: Classes and Objects, along with these inheritance and polymorphism these help us to create code that can be extended and easily maintainable.