**Q1. What is the relationship between classes and modules?**

* **Relationship**:
  + A **module** is a file containing Python code (e.g., functions, classes, variables).
  + A **class** is a blueprint for creating objects and is defined within a module.
  + Multiple classes can be defined in a single module.
  + Example:

# my\_module.py

class MyClass:

pass

**Q2. How do you make instances and classes?**

* **Creating a Class**:
  + Use the class keyword.
  + Example:

class MyClass:

pass

* **Creating an Instance**:
  + Call the class as if it were a function.
  + Example:

instance = MyClass()

**Q3. Where and how should class attributes be created?**

* **Class Attributes**:
  + Defined directly within the class, outside any method.
  + Shared by all instances of the class.
  + Example:

class MyClass:

class\_attr = 42 # Class attribute

**Q4. Where and how are instance attributes created?**

* **Instance Attributes**:
  + Defined within methods (typically \_\_init\_\_) using self.
  + Unique to each instance.
  + Example:

class MyClass:

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

self.instance\_attr = value # Instance attribute

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

* **self**:
  + Refers to the instance of the class.
  + Used to access instance attributes and methods.
  + Example:

class MyClass:

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

self.value = value

**Q6. How does a Python class handle operator overloading?**

* **Operator Overloading**:
  + Achieved by defining special methods (e.g., \_\_add\_\_, \_\_sub\_\_).
  + Example:

class MyClass:

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

self.value = value

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

return MyClass(self.value + other.value)

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

* **When to Use**:
  + When the class represents a mathematical or logical entity (e.g., vectors, matrices).
  + When the operators have a clear and intuitive meaning for the class.

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

* **Most Popular Form**:
  + Arithmetic operators (e.g., +, -, \*, /).
  + Example:

class Vector:

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

self.x = x

self.y = y

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

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

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

1. **Inheritance**:
   * Allows a class to inherit attributes and methods from another class.
   * Promotes code reuse and hierarchy.
2. **Polymorphism**:
   * Enables objects of different classes to be treated as objects of a common superclass.
   * Allows flexibility and dynamic behavior.