Q1. What is the relationship between classes and modules?

Sol. Classes are always nested inside a module; they are attributes of a module object. Classes and modules are both namespaces, but classes correspond to statements (not entire files) and support the OOP notions of multiple instances, inheritance, and operator overloading. In a sense, a module is like a single-instance class, without inheritance, which corresponds to an entire file of code.

Q2. How do you make instances and classes?

Sol. Classes are made by running class statements; instances are created by calling aclass as though it were a function.

Q3. Where and how should be class attributes created?

Sol. Class attributes are created by assigning attributes to a class object. They are normally generated by top-level assignments nested in a class statement—each name assigned in the class statement block becomes an attribute of the class object (technically, the class statement scope morphs into the class object’s attribute (namespace). Class attributes can also be created, though, by assigning attributes to the class anywhere a reference to the class object exists—i.e., even outside the class statement.

Q4. Where and how are instance attributes created?

Sol. Instance attributes are created by assigning attributes to an instance object. They are normally created within class method functions inside the class statement by assigning attributes to the self argument (which is always the implied instance). Again, though, they may be created by assignment anywhere a reference to the instance appears, even outside the class statement. Normally, all instance attributes are initialized in the \_\_init\_\_ constructor method; that way, later method calls can assume the attributes already exist.

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

Sol. self is the name commonly given to the first (leftmost) argument in a class method function; Python automatically fills it in with the instance object that is the implied subject of the method call. This argument need not be called self (though this is a very strong convention); its position is what is significant.

Q6. How does a Python class handle operator overloading?

Sol. Operator overloading is coded in a Python class with specially named methods;they all begin and end with double underscores to make them unique. These arenot built-in or reserved names; Python just runs them automatically when an instanceappears in the corresponding operation. Python itself defines the mappingsfrom operations to special method names.

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

Sol. Operator overloading is useful to implement objects that resemble built-in types e.g., sequences or numeric objects such as matrixes), and to mimic the built-in type interface expected by a piece of code. Mimicking built-in type interfaces enables you to pass in class instances that also have state information—i.e., attributes that remember data between operation calls. You shouldn’t use operator overloading when a simple named method will suffice, though.

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

Sol. The \_\_init\_\_ constructor method is the most commonly used; almost every classuses this method to set initial values for instance attributes and perform other

startup tasks.

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

Sol. The special self argument in method functions and the \_\_init\_\_ constructor method are the two cornerstones of OOP code in Python.