**Assignment - 19**

Q1. Define the relationship between a class and its instances. Is it a one-to-one or a one-to-many partnership, for example?

Ans: The relationship between a class and its instances is a one-to-many partnership. A class serves as a blueprint or template for creating multiple instances, each representing a specific object with its own unique attributes and behaviors.

Q2. What kind of data is held only in an instance?

Ans: Data held only in an instance includes instance variables, which are unique to each instance and store specific information or state related to that instance.

Q3. What kind of knowledge is stored in a class?

Ans: Knowledge stored in a class includes class variables, which are shared among all instances of the class, as well as methods, which define the behaviors and actions that instances of the class can perform.

Q4. What exactly is a method, and how is it different from a regular function?

Ans: A method is a function that is defined within a class and operates on instances of that class. The main difference between a method and a regular function is that a method is associated with a class and can access and modify the attributes of instances of that class.

Q5. Is inheritance supported in Python, and if so, what is the syntax?

Ans: Yes, inheritance is supported in Python. The syntax for inheritance is to specify the parent class(es) inside parentheses after the class name when defining a new class. For example:

class Subclass(ParentClass):

# subclass definition

Q6. How much encapsulation (making instance or class variables private) does Python support?

Ans: Python supports encapsulation by allowing the use of private variables and methods using the naming convention of prefixing them with a double underscore (\_\_). However, Python does not enforce strict encapsulation like some other languages do.

Q7. How do you distinguish between a class variable and an instance variable?

Ans: A class variable is shared among all instances of the class and is defined within the class itself. An instance variable is specific to each instance of the class and is defined within the methods of the class or assigned to instances directly.

Q8. When, if ever, can self be included in a class's method definitions?

Ans: The self parameter can be included in a class's method definitions to access and modify instance variables and invoke other methods within the class. It must be included as the first parameter in every method definition within the class.

Q9. What is the difference between the \_ \_add\_ \_ and the \_ \_radd\_ \_ methods?

Ans: The \_\_add\_\_ method is called when an object is added to another object using the + operator, while the \_\_radd\_\_ method is called when the right-hand object does not support the operation, and Python needs to try the reversed operation (i.e., adding the left-hand object to the right-hand object).

Q10. When is it necessary to use a reflection method? When do you not need it, even though you support the operation in question?

Ans: Reflection methods, such as \_\_getattr\_\_ and \_\_setattr\_\_, are necessary when you want to intercept attribute access or assignment and customize the behavior. You do not need reflection methods if you don't require such custom behavior.

Q11. What is the \_ \_iadd\_ \_ method called?

Ans: The \_\_iadd\_\_ method is called when the += operator is used to perform in-place addition on an object. It is a special method used for implementing the augmented assignment operator.

Q12. Is the \_ \_init\_ \_ method inherited by subclasses? What do you do if you need to customize its behavior within a subclass?

Ans: Yes, the \_\_init\_\_ method is inherited by subclasses. If you need to customize its behavior within a subclass, you can override it by defining a new \_\_init\_\_ method in the subclass. To customize the behavior while still utilizing the superclass's \_\_init\_\_ method, you can call super().\_\_init\_\_() within the subclass's \_\_init\_\_ method.