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: It's a one-to-many relationship where a class serves as a blueprint, defining the structure and behavior of multiple instances (objects). Each instance is a unique instantiation of that class, possessing its own set of attributes and behaviors.

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

Ans: Instance-specific data, also known as instance variables, are held exclusively within an instance. These variables capture unique characteristics or states that vary from one instance to another.

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

Ans: Classes encapsulate shared characteristics and behaviors among instances. They hold class variables, which are attributes shared by all instances of the class, as well as methods defining the behavior and operations applicable to those instances.

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

Ans: A method is a function defined within a class and is associated with instances of that class. The key difference lies in the method's ability to access and operate on the instance's data through the self parameter, which represents the instance itself.

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

Ans: Yes, Python supports inheritance. The syntax for inheritance involves specifying the parent class(es) inside parentheses after the subclass name in the class definition. For instance:

class SubClassName(ParentClass1, ParentClass2, ...):

# Class body

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

Ans: Python supports encapsulation to a certain extent by convention. It utilizes name mangling (preceding variables with double underscores \_\_) to make instance variables private, although this isn't a strict enforcement of access restrictions.

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

Ans: Class variables are shared among all instances of the class and are accessed using the class name. Instance variables are specific to each instance and are accessed through the instance itself using self.

Q8. When, if ever, can self be included in a class&#39;s method definitions?

Ans: 'self' is included as the first parameter in method definitions within a class, representing the instance calling the method. It's needed to access instance-specific attributes and methods.

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

Ans: \_ \_add\_ \_ is used for the addition operation (+) when the object is on the left side, while \_ \_radd\_ \_ is called when the object is on the right side of the addition operation.

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, like hasattr() or getattr(), are required when you need to dynamically inspect or manipulate attributes or methods of an object. You may not need them when the attributes or methods are known and can be accessed directly.

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

Ans: \_ \_iadd\_ \_ is called for the += operation and is part of the augmented assignment operators.

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. To customize its behavior in a subclass, you can override it by defining a new \_ \_init\_ \_ method in the subclass, while optionally calling the parent class's \_ \_init\_ \_ method within the subclass's method using super().