

Python_advance_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?

In []: Ans: Relationship between a **class** and its instances **is** a one to many partnership.

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

In []: Ans: Instance objects contains the Instance variables which are specific to that specific Instance object.

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

In []: Ans: Class creates a user-defined data structure, which holds its own data members **and** member functions, which can be accessed **and** used by creating an instance of that class. A **class** is like a blueprint **for** an object.

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

In []: Ans: The methods **with** a can be used to access the insatnce variables of its instance.
So, the object's **state can be modified by its method**. Function can't access the attributes of an instance of a **class** or can't **modify the state of the object**.

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

In []: Ans: Yes, Python supports inheritance. The Types of Inheritance Supported by Python are

Simple Inheritance
Multiple Inheritance
Multilevel lInheritance
Hybrid Inheritance
Hierracial Inheritance

In []: **class** Person:
 def __init__(self, fname, lname):
 self.first_name=fname
 self.last_name=lname
class Student(Person):
 pass

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

In []: Ans: Encapsulation describes the idea of wrapping data **and** the methods that work on data within one unit.
This puts restrictions on accessing variables **and** methods directly **and** can prevent the accidental modification of data.
To prevent accidental change, an objects variable can only be changed by an objects method.

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

In []: Ans: The Class Attribute **is** available to all the instance objects of that class.
whereas Instance
Attributes are accessible only to the object **or** Instance of that
class.

A single copy of Class attributes **is** maintained by pvm at the **class** level.
Whereas difference copies of instance attributes are maintained by pvm at objects/
instance level.

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

In []: Ans: Yes, self can included **in class** method definations to access the instance variable inside **class** methods.

Q9. What is the difference between the add and the radd methods ?

In []: Ans: Entering __radd__ Python will first try __add__(), **and if** that returns Not Implemented Python will check **if** the right-hand operand implements __radd__, **and if** it does, it will call __radd__() rather than raising a TypeError

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

In []: Ans: Reflection method we often encounter the requirement that a method **in** the executing object, **or** a variable **in** the calling object, **or** a field of the object should be assigned, **while** the method name **or** field name can **not** be determined when encoding the code, **and** need to be input **in** the form of passing strings through parameters.

Q11. What is the iadd method called?

In []: Ans: __iadd__ method **is** called when we use implementation like a+=b which **is** a.__iadd__(b)

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

In []: Ans: Yes, __init__ method will be inherited by subclasses. **if** we want to customize its behaviour within a subclass we can use super() method.