Python Advanced 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: Relationship between a class and its instances is a one to many partnership. Object is a physical entity or real word entity. An object occupies space in memory. Example:-banana,chair,pen etc are objects. Class is a logical entity which does not exist in real world. Or we can say that class is a blueprint to create objects of similiar types. A class does not take space in memory. Examples:-fruits,furniture,vehicles etc.

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

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?

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. Class is a logical entity which does not exist in real world Or we can say that class is a blueprint to create objects of similiar types. A class does not take space in memory. Examples:-fruits,furniture,vehicles etc.

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

Ans: The methods with a class 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. A function is independent, whereas a method is a function linked with an object. Explicit data is passed on to a function, whereas a method completely passes the object on which it was called in the program. A method is Object-oriented programming while a function has standalone functionality

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

Ans: Yes,Python supports inheritance. The Types of Inheritence Supported by Python are:

1. Simple Inheritence
2. Multiple Inheritence
3. Multilevel lInheritence
4. Hybrid Inheritence
5. Hierracial Inheritence

```
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?

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?

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 different 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?

Ans: Yes, self can be included in class method definitions to access the instance variables inside class methods.

Q9. What is the difference between the ***__add__*** *and the* ***__radd__*** *methods ?*

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**.

The expression a+b is internally translated to the method call a.__add__(b). But if a and b are of different types, it is possible that a's implementation of addition cannot deal with objects of b's type (or maybe a does not have a **add** method, at all). So, if a.__add__(b) fails, Python tries b.__radd__(a) instead, to see if b's implementation can deal with objects of a's type.

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 refers to the ability for code to be able to examine attributes about objects that might be passed as parameters to a function. For example, if we write type(obj) then Python will return an object which represents the type of obj. Using reflection, we can write one recursive reverse function that will work for strings, lists, and any other sequence that supports slicing and concatenation. If an obj is a reference to a string, then Python will return the str type object. Further, if we write str() we get a string which is the empty string. In other words, writing str() is the same thing as writing “”. Likewise, writing list() is the same thing as writing [].

```
x = 5

def testFunction():
  print("Test")

y = testFunction

if (callable(x)):
    print("x is callable")
else:
    print("x is not callable")
```

```
if (callable(y)):
    print("y is callable")
else:
    print("y is not callable")
```

```
x is not callable
y is callable
```

Q11. What is the __iadd__ method called?

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 ?

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