Assignment 19 Solutions

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: A class is a blueprint which you use to create objects. An object is an instance of a class - it's a concrete 'thing' that you made using a specific class. So, 'object' and 'instance' are the same thing, but the word 'instance' indicates the relationship of an object to its class. a single object of a class can access multille functions defiend in the class in form of ine to many relationships.

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 functionalit

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 IInheritence
4. Hybrid Inheritence
5. Hierracial Inheritence

In [1]:

```
class Person:
    def __init__(self, fname, lname):
        self.first_name = fname
        self.last_name = lname
class Student(Person):
    pass
```

In [2]:

```
#single level inheritence
class Ineuron:
    company_website = 'https://ineuron.ai/'
    name = 'iNeuron'

    def contact_details(self):
        print('Contact us at ', self.company_website)


class Datascience(Ineuron):
    def __init__(self):
        self.year_of_establishment= 2018

    def est_details(self):
        print('{0} Company was established in {1}'
              .format(self.name,self.year_of_establishment))


ds = Datascience()
ds.est_details()
```

```
iNeuron Company was established in 2018
```

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

Ans: We can protect variables in the class by marking them private. To define a private variable we can add two underscores as a prefix at the start of a variable

name. Private members are accessible only within the class, and we can't access them directly from the class objects.

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

Ans: `Class Attribute` : It usually maintains a single shared value for all instances of class even if no instance object of the class exists.

`Instance Attribute` : It usually reserves memory for data that the class needs.

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: Self is always pointing to Current Object.

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 [].

In [4]:

```
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)`**

In [5]:

```
class NumString:

    def __init__(self, value):
        self.value = str(value)

    def __int__(self):
        return int(self.value)

    def __str__(self):
        return self.value

    def __add__(self, other):
        return int(self) + other

    def __iadd__(self, other):
        self.value = self + other
        return self.value

some_num1 = NumString(0) # Create an instance of NumString with 0 as value.
print(str(some_num1)) # Check what is contained within self.value in this instance.

some_num1 + 1 # We add some_num1 to 1, this returns back the calculation using the __add__ method
print(str(some_num1)) # check what some_num1 is holding for a self.value

some_num1 += 2 # We in-place add some_num1 to 2, this stores and returns the new stored value using __iadd__ method.
print(str(some_num1)) # check what some_num1 is holding now?

```

```
0
0
2
```

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

Ans: In Python, it is not compulsory that parent class constructor will always be called first. The order in which the `init` method is called for a parent or a child class can be modified.

A subclass can do more than that; it can define a method that has exactly the same method signature (name and argument types) as a method in its `superclass`. In that case, the method in the subclass overrides the method in the `superclass` and effectively replaces its implementation