

python_advance_assignment3

1. What is the concept of an abstract superclass?

Ans:Abstract class/superclass can be considered as a blueprint for other classes.s:s abstractclass/superclass can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. Whereas an abstract method is a method that has a declaration but does not have an implementation.

In []:

```
from abc import ABC, abstractmethod
class Polygon(ABC): # Abstract Class
    @abstractmethod
    def noofsides(self): # Abstract Method
        pass
class Triangle(Polygon):
    def noofsides(self): # overriding abstract method in child class Triangle
        print("I have 3 sides")
class Pentagon(Polygon):
    def noofsides(self): # overriding abstract method in child class Pentagon
        print("I have 5 sides")
```

2. What happens when a class statement’s top level contains a basic assignment statement?

Ans:An abstract class/superclass can be considered as a blueprint for other classes. When a Class statement's top level contains a basic assignment statement, it is usually treated as a class attribute or class level variable.

where as assignment statements inside methods are treated as instance attributes or local attributes.

When an instance of a class is created a single copy of class attributes is maintained and shared to all instances of class. where as each instance object maintains its own copy of instance variables.

In []:

```
class Person:
    species = 'Homesapiens' # class attribute
    def __init__(self,name,gender):
        self.name = name # instance attributes
        self.gender = gender
```

3. Why does a class need to manually call a superclass’s init method?

Ans:An abstract class/superclass can be considered as a blueprint for other classes. If a child class has init method, then it will not inherit the **init** method of the parent class. In other words the init method of the child class overrides the **init** method of the parent class. So we have to manually call a parent superclass's init using super() method.

In [1]:

```
class Person:
    def __init__(self,name,age):
        self.name = name
        self.age = age
class Employee(Person):
    def __init__(self,name,age,salary):
        super().__init__(name,age)
        self.salary = salary
emp_1 = Employee('Suman',24,50000)
print(emp_1.__dict__)
```

{'name': 'Suman', 'age': 24, 'salary': 50000}

4. How can you augment, instead of completely replacing, an inherited method?

Ans:super() method can be used to augment, instead of completely replacing, an inherited method
class Person:

In [2]:

```
class Person:
    def __init__(self,name,gender):
        self.name = name
        self.gender = gender
class Employee(Person):
    def __init__(self,name,gender,salary):
        super().__init__(name,gender)
        self.salary = salary
emp_1 = Employee('Suman','Male',30000)
print(emp_1.__dict__)
```

{'name': 'Suman', 'gender': 'Male', 'salary': 30000}

5. How is the local scope of a class different from that of a function?

In []:

Ans: A Variable which **is** defined inside a function **is** local to that function. It **is** accessible **from** the point at which it **is** defined until the end of the function, **and** exists **for as long as** the function **is** existing. Similarly a variable inside of a **class** also has a local variable scope. Variables which are defined **in** the **class** body (but outside all methods) are called **as class** level variables **or class** attributes. They can be referenced by their bare names within the same scope, but they can also be accessed **from** outside this scope **if** we use the attribute access operator (.) on a **class** or an instance of the class.

In [5]:

```
def hello(name):
    name = name
    print(f'you\'re name is{name}')
    hello('Kakon De')
try:
    name
except NameError:
    print('Name variable is not available outside hello function scope')
class Person:
    species = "HomeSapines"
    def __init__(self):
        pass
print(Person.species) # Accessing species using class name
Male = Person()
print(Male.species) # Accessing species using instance of class
```

Name variable is not available outside hello function scope
HomeSapines
HomeSapines