K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)

Department of Science and Humanities

Batch: B2	**Roll No.: 16010121110**
Experiment / assignment / tutorial No.	
Grade: AA / AB / BB / BC / CC / CD /DD	

TITLE: Inheritance

AIM: Write a program to implement inheritance to display information of bank account.

Expected OUTCOME of Experiment: Apply Object oriented programming concepts in Python

Resource Needed: Python IDE

Theory:

Inheritance is the capability of one class to derive or inherit the properties from some another class. The benefits of inheritance are:

1. It represents real-world relationships well.

2. It provides reusability of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.

3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Syntax:

```
class Person(object):

    # Constructor
    def __init__(self, name):
        self.name = name
    # Inherited or Sub class (Note Person in bracket)
    class Employee(Person):


    # Here we return true
    def isEmployee(self):
        return True
```

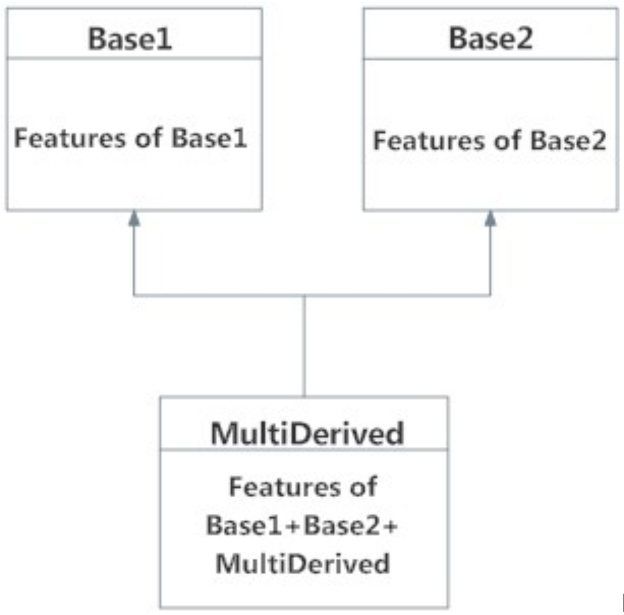
Different forms of Inheritance:

1. Single inheritance: When a child class inherits from only one parent class, it is called as single inheritance. We saw an example above.

2. Multiple inheritance: When a child class inherits from multiple parent classes, it is called as multiple inheritance.

```
class Base1(object):
 . . . .
class Base2(object):
 . . . .

class Derived(Base1, Base2):
```

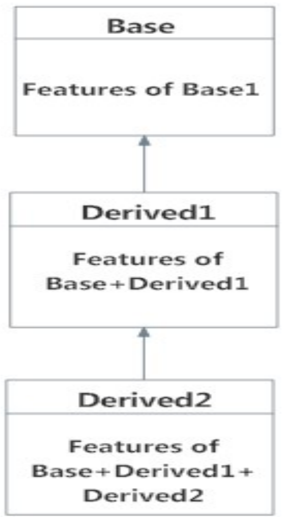
. . . .



Multiple Inheritance in Python

3. **Multilevel inheritance**: When we have child and grand child relationship.

```
class Person(object):

      . . .

# Inherited or Sub class (Note Person in bracket)

class Child(Base):

      . . .

# Inherited or Sub class (Note Child in bracket)

class GrandChild(Child):

    . . . .
```


Base
Features of Base1
Derived1
Features of Base+Derived1
Derived2
Features of Base+Derived1+ Derived2

Multilevel Inheritance

Private members of parent class:

Python doesn't have any mechanism that effectively restricts access to any instance variable or method. Python prescribes a convention of prefixing the name of the variable/method with single or double underscore to emulate the behaviour of protected and private access specifiers.

We don’t always want the instance variables of the parent class to be inherited by the child class i.e. we can make some of the instance variables of the parent class private, which won’t be available to the child class.

All members in a Python class are public by default. Any member can be accessed from outside the class environment.

Example: Public Attributes

```
class employee:
   def __init__(self, name, sal):
       self.name=name
       self.salary=sal
e1=  employee(1000)
print(e1.salary)
```

Python's convention to make an instance variable protected is to add a prefix _ (single underscore) to it. This effectively prevents it to be accessed, unless it is from within a sub-class. This doesn't prevent instance variables from accessing or modifying the instance

Example: Protected Attributes

```
class employee:
   def __init__(self, name, sal):
       self._name=name  # protected attribute
       self._salary=sal # protected attribute
```

A double underscore __ prefixed to a variable makes it private. It gives a strong suggestion not to touch it from outside the class. Any attempt to do so will result in an AttributeError:

Example: Private Attributes

```
class employee:
   def __init__(self, name, sal):
    self.__name=name  # private attribute
    self.__salary=sal # private attribute
```

Python performs name mangling of private variables. Every member with double underscore will be changed to _object._class__variable. If so required, it can still be accessed from outside the class, but the practice should be refrained.

```
e1=Employee("Bill",10000)
print(e1._Employee__salary)
e1._Employee__salary=20000
print(e1._Employee__salary)
```

super() method and method resolution order(MRO)

In Python, super() built-in has two major use cases:

- Allows us to avoid using base class explicitly
- Working with Multiple Inheritance

super() with Single Inheritance:

In case of single inheritance, it allows us to refer base class by super().

```
class Mammal(object):
  def __init__(self, mammalName):
    print(mammalName, 'is a warm-blooded animal.')

class Dog(Mammal):
  def __init__(self):
    print('Dog has four legs.')
    super().__init__('Dog') #  instead of Mammal.__init__(self, 'Dog')


d1 = Dog()
```

The super() builtin returns a proxy object, a substitute object that has ability to call method of the base class via delegation. This is called indirection (ability to reference base object with super())

Since the indirection is computed at the runtime, we can use point to different base class at different time (if we need to).

Method Resolution Order (MRO):

It's the order in which method should be inherited in the presence of multiple inheritance. You can view the MRO by using __mro__ attribute.

Problem Definition:
1. For given program find output

Sr.No	Program	Output

1	class Rectangle:
	    def __init__(self, length, width):
	        self.length = length
	        self.width = width
	
	    def area(self):
	        return self.length * self.width
	
	    def perimeter(self):
	        return 2 * self.length + 2 * self.width
	
	class Square(Rectangle):
	    def __init__(self, length):
	        super().__init__(length, length)
	
	square = Square(4)
	print(square.area())
		16

2	class Person:
	  def __init__(self, fname, lname):
	    self.firstname = fname
	    self.lastname = lname
	
	  def printname(self):
	    print(self.firstname, self.lastname)
	
	class Student(Person):
	  def __init__(self, fname, lname, year):
	    super().__init__(fname, lname)
	    self.graduationyear = year
	
	x = Student("Wilbert", "Galitz", 2018)
	print(x.graduationyear)
		2018

3	class Base1(object):
	    def __init__(self):
	        self.str1 = "Python"
	        print("First Base class")
	
	class Base2(object):
	    def __init__(self):
	        self.str2 = "Programming"
	        print("Second Base class")
	
	class Derived(Base1, Base2):
	    def __init__(self):
	
	        # Calling constructors of Base1
	        # and Base2 classes
	        Base1.__init__(self)
	        Base2.__init__(self)
	        print("Derived class")
	
	    def printStrs(self):
	        print(self.str1, self.str2)
	
	ob = Derived()
	ob.printStrs()
		First Base class
		Second Base class
		Derived class
		Python Programming

2. Assume that a bank maintains two kinds of accounts for customers, one called as savings account and the other as current account. The savings account provides simple interest and withdrawal facilities but no cheque book facility. The current account provides cheque book facility but no interest. Current account holders should also maintain a minimum balance Rs. 500 and if the balance falls below this level, a service charge is imposed to 2%.

Create a class account that stores customer name, account number and type of account. From this derive the classes cur_acct and sav_acct to make them more specific to their requirements. Include necessary member functions in order to achieve the following tasks:

- Accept deposit from a customer and update the balance.
- Display the balance.
- Compute and deposit interest.
- Permit withdrawal and update the balance.
- Check for the minimum balance, impose penalty, necessary and update the balance.

Result

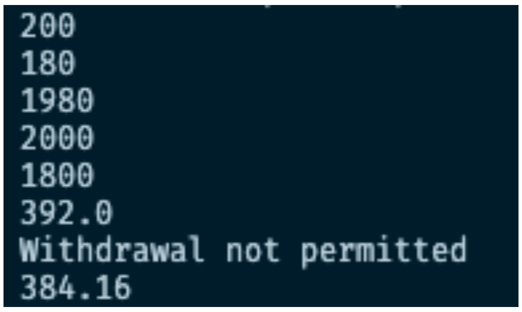
Books/ Journals/ Websites referred:

1. **https://github.com/Aatmaj-Zephyr/Learning-Python**
2. **Reema Thareja , "Python Programming: Using Problem Solving Approach", Oxford University Press, First Edition 2017, India**
3. **Sheetal Taneja and Naveen Kumar," Python Programing: A Modular Approach", Pearson India, Second Edition 2018, India**
4. https://www.programiz.com/python-programming/methods/built-in/super
5. https://www.tutorialsteacher.com/python/private-and-protected-access-modifiers-in-python
6. https://www.geeksforgeeks.org/inheritance-in-python/

Implementation details:

```
class account():
    #This is the superclass Account.
    def __init__(self):
        self.balance = 0
        pass
    def deposit(self,ammount):
        self.balance +=ammount
    def withdraw(self,ammount):
        if(self.balance > ammount):
            self.balance -=ammount
        else:
            print("Withdrawal not permitted")
    def display(self):
        print(self.balance)
class saving_account(account):
    #Initializing the class saving_account with the 
parameter rate_of_interest.
    def __init__(self,rate_of_interest):
        super().__init__()
        self.rate_of_interest = rate_of_interest
    #Calculating the interest on the balance.
    def deposit_interest(self):
        self.balance +=self.rate_of_interest*self.balance


class current_account(account):
    #Initializing the class current_account with the 
parameters minimum_balance and service_charge.
    def __init__(self,minimum_balance,service_charge):
        super().__init__()
        self.minimum_balance = minimum_balance
        self.service_charge = service_charge/100
    #The below code is defining a function withdraw 
which is a subclass of the superclass Account.
    def withdraw(self,amount):
        super().withdraw(amount)
        if(self.balance<=self.minimum_balance):
            self.penalty()
    #Calculating the penalty for the current account.
    def penalty(self):
        self.balance-=self.service_charge*self.balance

my_saving_account=saving_account(rate_of_interest=1
0) #10% interest rate
my_saving_account.deposit(200)
my_saving_account.display()
my_saving_account.withdraw(20)
my_saving_account.display()
my_saving_account.deposit_interest()
my_saving_account.display()


my_current_account=current_account(minimum_balanc
e=500,service_charge=2)
my_current_account.deposit(2000)
my_current_account.display()
my_current_account.withdraw(200)
my_current_account.display()
my_current_account.withdraw(1400)
my_current_account.display()
my_current_account.withdraw(1400)
my_current_account.display()
```

Output(s):



```
200
180
1980
2000
1800
392.0
Withdrawal not permitted
384.16
```

Conclusion:

In this experiment we have understood the working behind object oriented programming in python we understood the working of inheritance in python by creating different classes and sub-classes we also understood how to derive methods from the superclass object constructors in python can be created using a special function called as __init__ .We can use this knowledge to make better quality codes which are more flexible and dynamic in nature object oriented programming is a key tool for writing elegant code.

Post Lab Questions:

1. Explain *isinstance()* and *issubclass()* functions with example?

Isinstance is a function which helps to understand if a particular object is an instance of a particular class or not. In the same manner, is subclass checks if a class is a subclass of a superclass or not. Example we can use it to test if Tommy is an instance of dog using isinstance() function and we can check if class dog is a subclass of class animal or not using the issubclass() function.

Date:2 July 2022 **Signature of faculty in-charge**