Batch: A3	**Roll No.: 16010121051**
Experiment / assignment / tutorial No.	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

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.

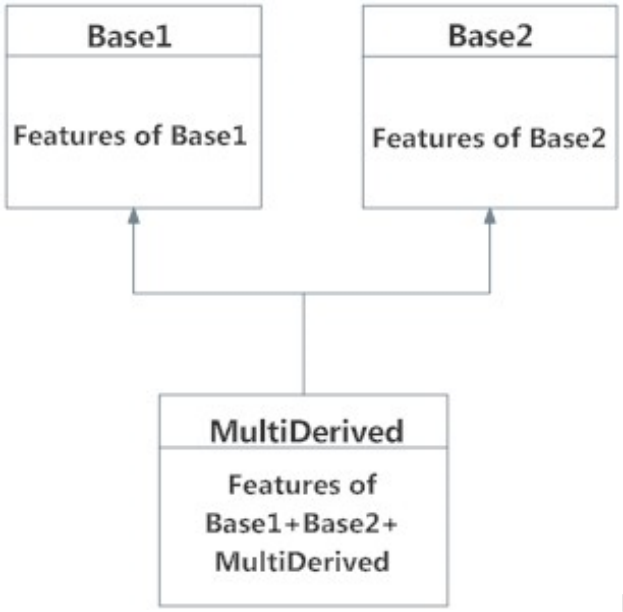
Syntax:

```
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.

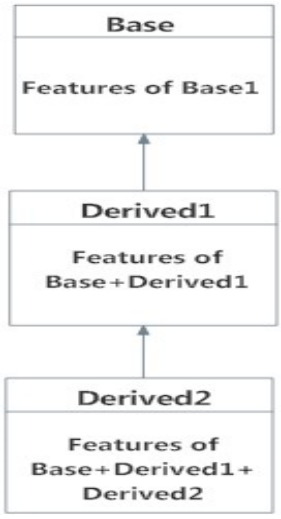
```
class Base1(object):
 . . . .
class Base2(object):
 . . . .

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):
   . . . .
```



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
		2018

	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)	

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

Implementation details:

```
class account:
    rate=0.05
    minimum_amount=500
    balance=0
    def __init__ (self,name,account_type,account_number):
        self.name=name
        self.account=account_type
        self.account_number=account_number
    def details(self):
```

```
        print("Name: ",self.name)
        print("Account Type: ",self.account_type)
        print("Account Number: ",self.account_number)
        print("Balance",self.balance)

class sav_acc(account):
    def withdraw(self,amount):

        if self.balance==0:
            print("Error can't withdraw account empty !!")
        else:
            self.balance-=amount
            print("The current balance is: ",self.balance)

    def deposit(self,amount):

        time=int(input("Enter the time you want to deposit: "))
        self.balance+=(self.balance+amount)*(1+self.rate*time)
        print("The current balance is: ",self.balance)

class curr_acc(account):
    def display(self):

        print("The current account only provides checkbook facility")
        self.balance = 500
        print(self.balance)

    def withdraw(self,amount):

        print(self.balance)

        if self.balance<=500:
            print("Your balance is only 500 .... if you withdraw then
service charge of 2% will be imposed on you!...")
            a = int(input("Press 1 to confirm"))
            if a==1:
                self.balance =self.balance-amount
                if((self.balance) <0):
                    print("Cant proceed as your balance is not
suffiecient")
                    self.balance = self.balance + amount
                else:
```

```
                    self.balance -= self.balance*0.02
                    print("The current balance is: ",self.balance)
            else:
                self.balance =self.balance-amount
                print("The current balance is: ",self.balance)

        else:
            self.balance-=amount
            print("The current balance is: ",(self.balance-
(self.balance*0.02)))

    def deposit(self,amount):

        self.balance += amount
        if(self.balance < 500):
            print("Your balance is lower than 500 please add more money
or else ervice charge of 2% will be imposed on you!...PLease enter 1 to
enter more money ")
            choice = int(input("Enter your choice"))
            if choice == 1:
                self.balance = self.balance - amount
                amount = int(input("Enter the amount you want to add"))
                print("The current balance is: ",self.balance+amount)
        else:
            print("The current balance is: ",self.balance)


name=input("Enter name: ")
account_number=input("Enter account number: ")
account_type=input("Enter type of account 'saving' or 'current': ")
if account_type=='saving':
    obj=sav_acc(name,account_type,account_number)
else:
    obj=curr_acc(name,account_type,account_number)
    obj.display()

while(True):
    task=input("Enter 'deposit' to deposit or 'withdraw' for
withdrawing:")
    if task=='deposit':
        amount=int(input("Enter amount: "))
        obj.deposit(amount)
```

```
    if task=='withdraw':
        amount=int(input("Enter amount: "))
        obj.withdraw(amount)
    if task == '0':
        break
```

Output(s):

```
PS C:\Users\Vishr\Documents\GitHub\Python_sem2> python -u "c:\Users\Vishr\Documents\GitHub\Python_sem2\Bank.py"
WELCOME TO THE BANK !
Press 1 to create an account
Press 2 to access your account
Press 3 to deposite money in your account
Press 4 to withdraw money from your account
Press 5 to calculate intrest in savings account
Press 6 to transfer
Press 7 to access admin

>> 1
Enter your name: Beetroot
>> 1
Enter your name: Sammod
Enter your account type: current
Deposit cash: 5010
Thank you for choosing our bank !
>> 6
Enter your name: Beetroot
Beetroot's savings account
Enter the recievers name: Sammod
Enter the ammount you want to tranfer: 4000
Beetroot's balance 3280.0
Sammod's balance 9010
>>
```

Conclusion:

We can conclude that we have learnt about inheritance in classes and the benefits of it. Use of different functions like super() , isinstance() and issubclass() within classes as well.

Post Lab Questions:

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

The isinstance() function returns True if the specified object is of the specified type, otherwise False. If the type parameter is a tuple, this function will return True if the object is one of the types in the tuple.

Code :-

```
x = isinstance(5, int)
print(x)

class Polygon:
  def __init__(polygonType):
    print('Polygon is a ', polygonType)

class Triangle(Polygon):
  def __init__(self):

    Polygon.__init__('triangle')

print(issubclass(Triangle, Polygon))
print(issubclass(Triangle, list))
```

output:-

```
Install the latest PowerShell for new features and impro

PS C:\PythonProgramming> & "C:/Program Files/Python310/p
True
True
False
PS C:\PythonProgramming>
```

Date: _____ **Signature of faculty in-charge**