Batch: G3 Roll No.: 16010421063

Experiment / assignment / tutorial No. 7

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:

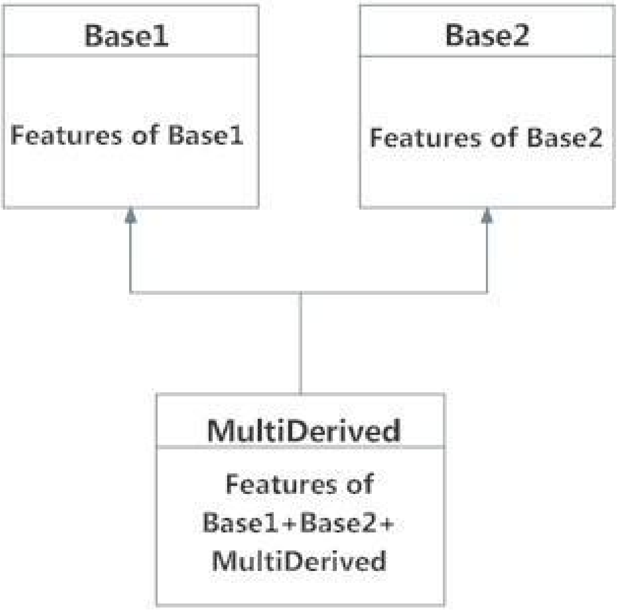
```
class Person(object):
  # Constructor
  def __init__(self, name):
    self.name = name
  # Inherited or Sub class (Note Person in bracket)
  class Employee(Person):

  # 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.

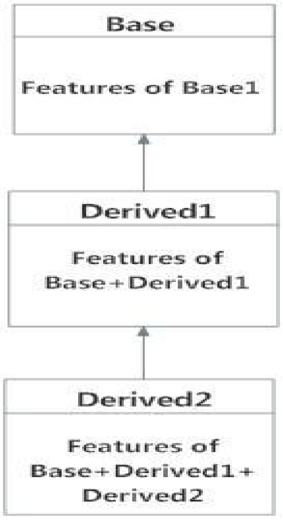
```
class Base1(object):
 . . . .
class Base2(object):
 . . . .

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):
    . . . .
```


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).

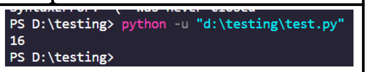
Method Resolution Order (MRO):

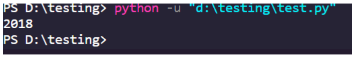
It's the order in which method should be inherited in the presence of multiple inheritance. You can view the MRO by using __mro__ attribute.

Problem Definition:

1. For given program find output

Sr.No	Program	Output

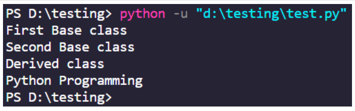
1	class Rectangle:
	    def __init__(self, length, width):
	        self.length = length
	        self.width = width
	
	    def area(self):
	        return self.length * self.width
	
	    def perimeter(self):
	        return 2 * self.length + 2 * self.width
	
	class Square(Rectangle):
	    def __init__(self, length):
	        super().__init__(length, length)
	
	square = Square(4)
	print(square.area())
		
		SyntaxError: '(' was never closed
		PS D:\testing> python -u "d:\testing\test.py"
		16
		PS D:\testing>

2	class Person:
	  def __init__(self, fname, lname):
	    self.firstname = fname
	    self.lastname = lname
		
		PS D:\testing> python -u "d:\testing\test.py"
		2018
		PS D:\testing>

```
  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()
```



```
PS D:\testing> python -u "d:\testing\test.py"
First Base class
Second Base class
Derived class
Python Programming
PS D:\testing>
```

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:

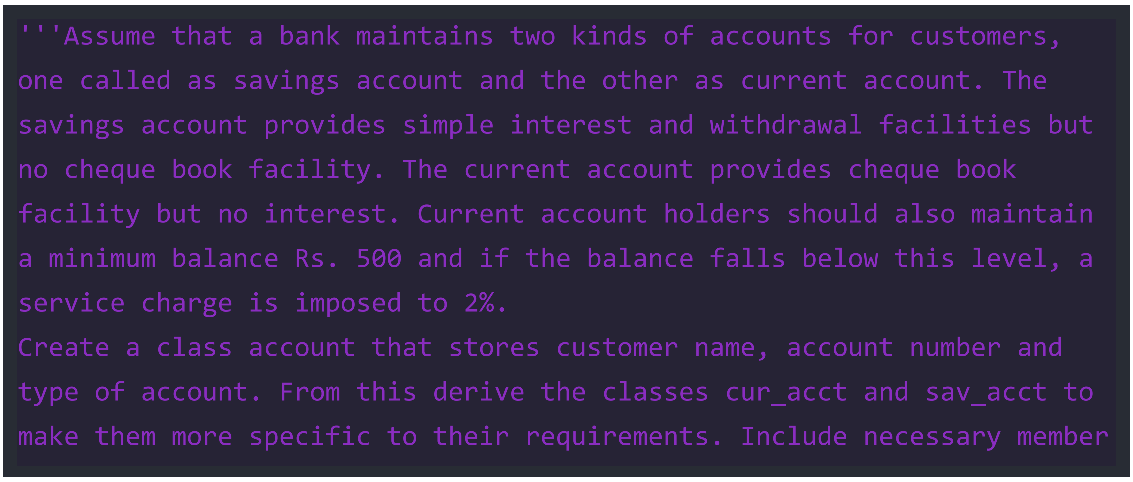
- Accept deposit from a customer and update the balance.
- Display the balance.
- Compute and deposit interest.
- Permit withdrawal and update the balance.
- Check for the minimum balance, impose penalty, if necessary and update the balance.

Result

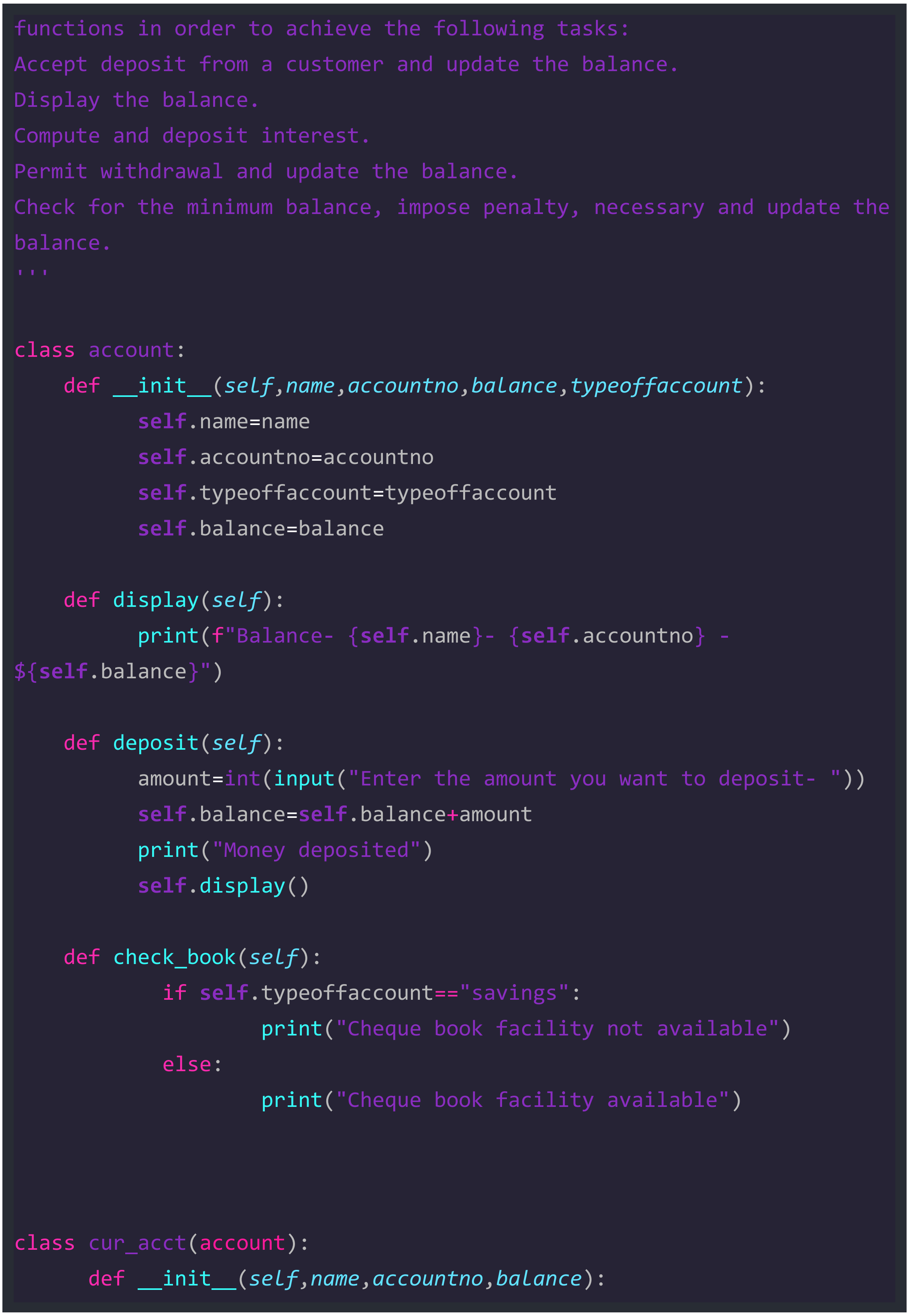
Books/ Journals/ Websites referred:

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/

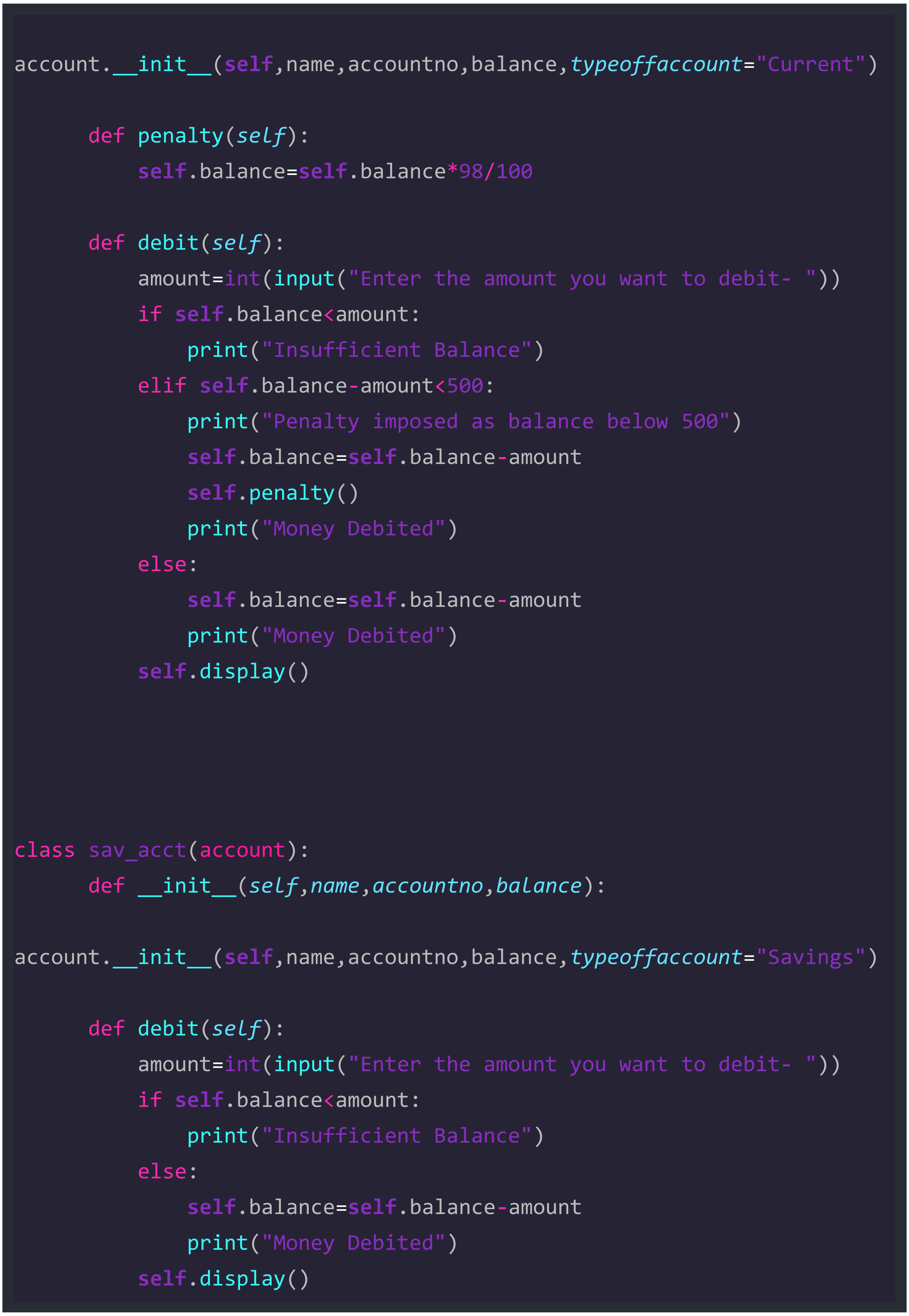
Implementation details:



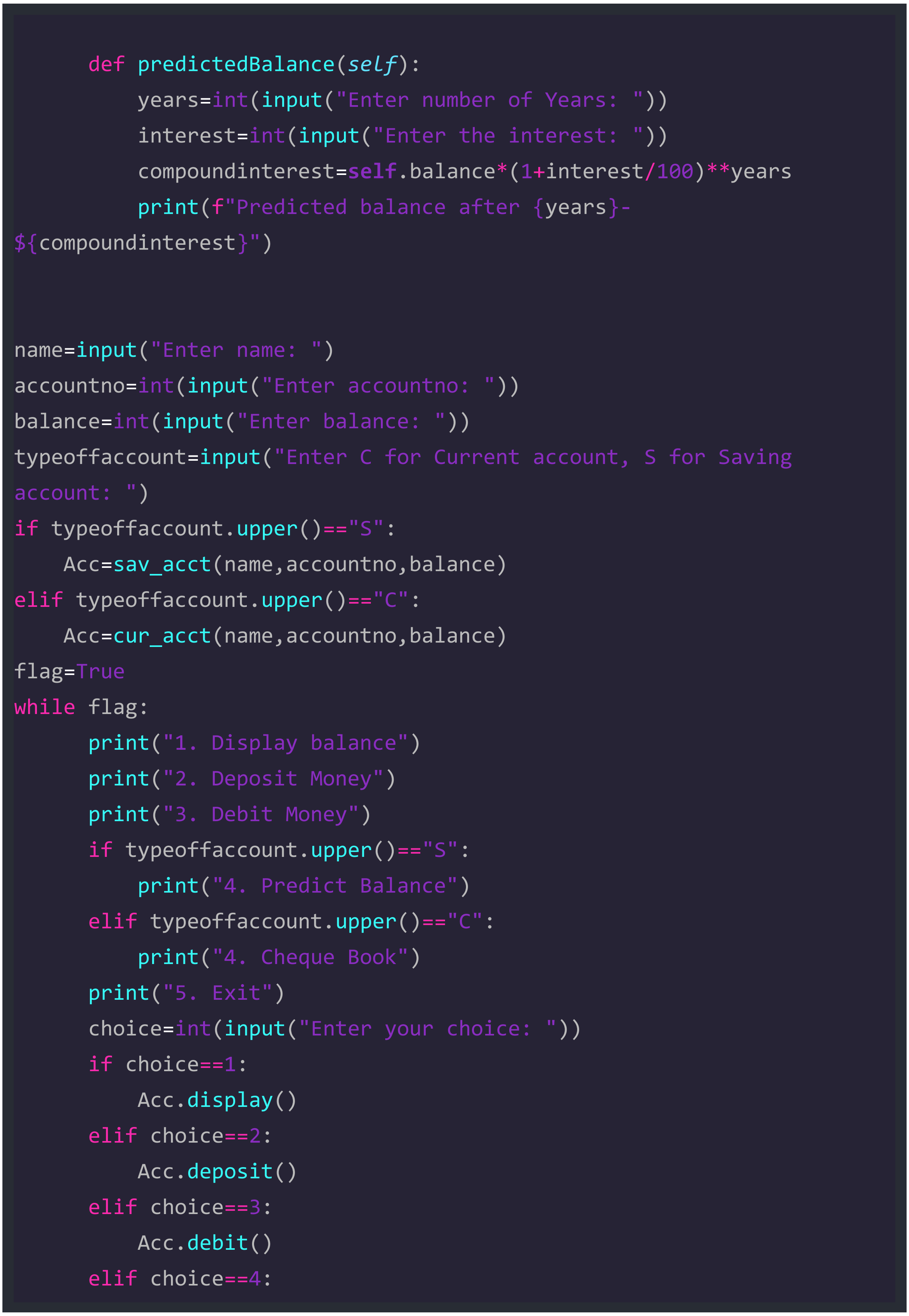
```
'''Assume that a bank maintains two kinds of accounts for customers,
one called as savings account and the other as current account. The
savings account provides simple interest and withdrawal facilities but
no cheque book facility. The current account provides cheque book
facility but no interest. 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
```



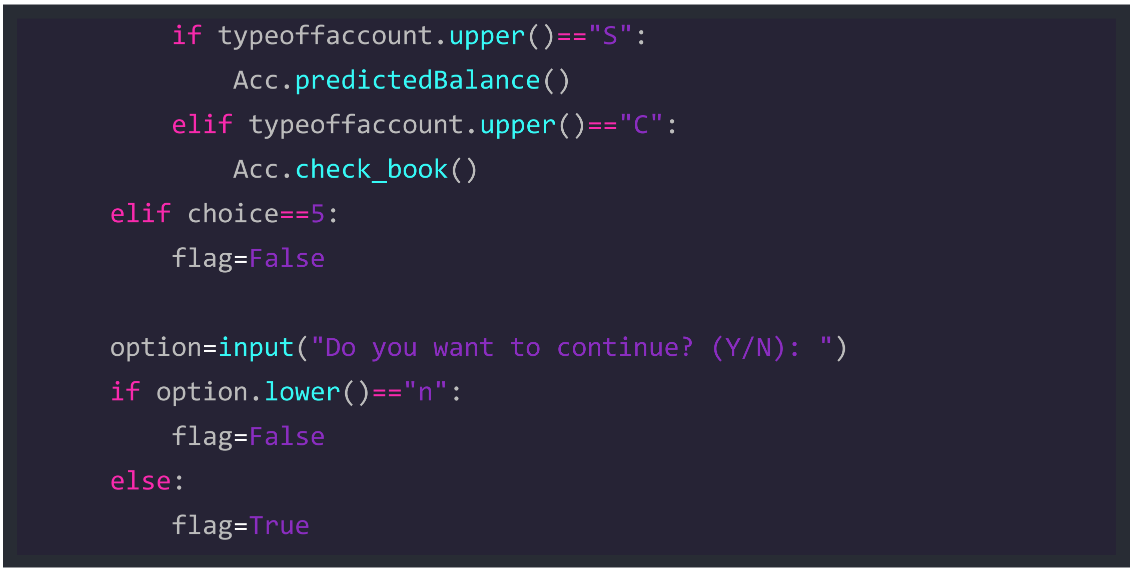
```
functions in order to achieve the following tasks:
Accept deposit from a customer and update the balance.
Display the balance.
Compute and deposit interest.
Permit withdrawal and update the balance.
Check for the minimum balance, impose penalty, necessary and update the
balance.
'''


class account:
    def __init__(self,name,accountno,balance,typeoffaccount):
          self.name=name
          self.accountno=accountno
          self.typeoffaccount=typeoffaccount
          self.balance=balance

    def display(self):
          print(f"Balance- {self.name}- {self.accountno} -
${self.balance}")

    def deposit(self):
          amount=int(input("Enter the amount you want to deposit- "))
          self.balance=self.balance+amount
          print("Money deposited")
          self.display()

    def check_book(self):
            if self.typeoffaccount=="savings":
                    print("Cheque book facility not available")
            else:
                    print("Cheque book facility available")



class cur_acct(account):
      def __init__(self,name,accountno,balance):
```



```
account.__init__(self,name,accountno,balance,typeoffaccount="Current")

      def penalty(self):
          self.balance=self.balance*98/100

      def debit(self):
          amount=int(input("Enter the amount you want to debit- "))
          if self.balance<amount:
              print("Insufficient Balance")
          elif self.balance-amount<500:
              print("Penalty imposed as balance below 500")
              self.balance=self.balance-amount
              self.penalty()
              print("Money Debited")
          else:
              self.balance=self.balance-amount
              print("Money Debited")
          self.display()




class sav_acct(account):
      def __init__(self,name,accountno,balance):

account.__init__(self,name,accountno,balance,typeoffaccount="Savings")

      def debit(self):
          amount=int(input("Enter the amount you want to debit- "))
          if self.balance<amount:
              print("Insufficient Balance")
          else:
              self.balance=self.balance-amount
              print("Money Debited")
          self.display()
```

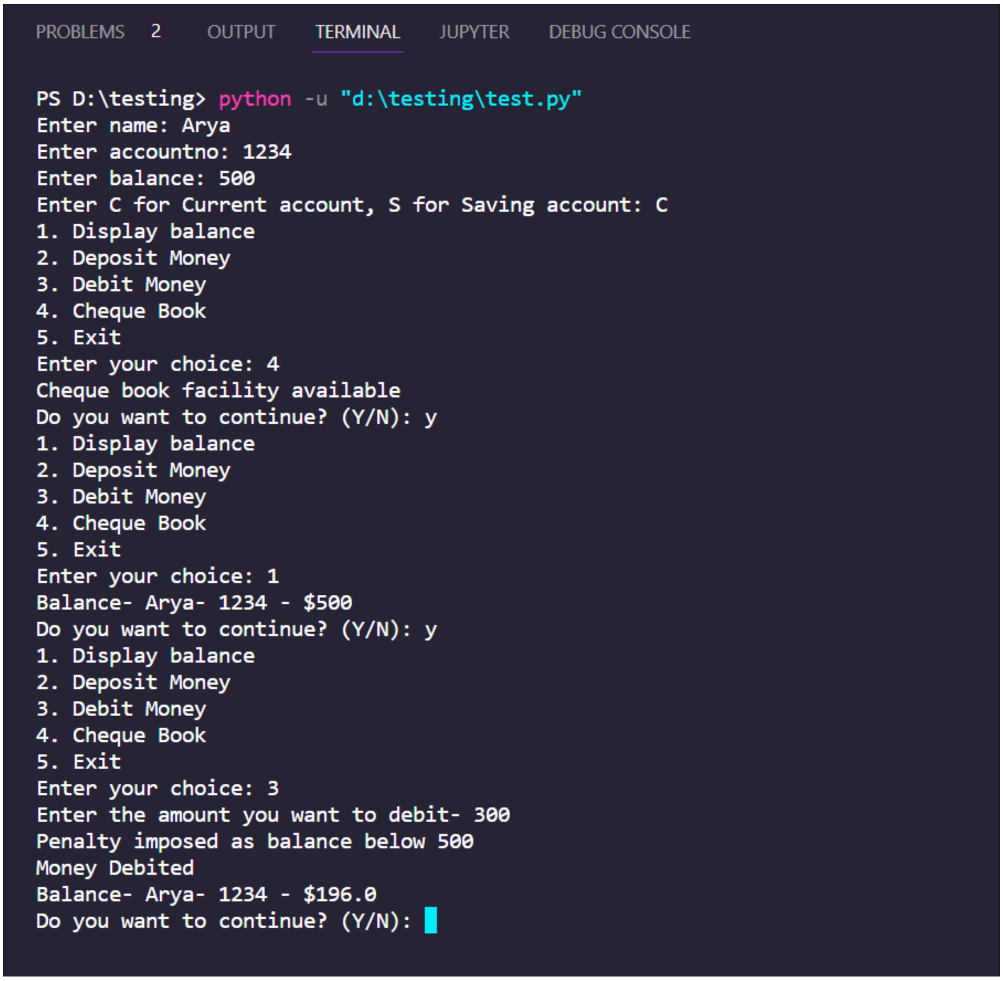


```
   def predictedBalance(self):
     years=int(input("Enter number of Years: "))
     interest=int(input("Enter the interest: "))
     compoundinterest=self.balance*(1+interest/100)**years
     print(f"Predicted balance after {years}-
${compoundinterest}")


name=input("Enter name: ")
accountno=int(input("Enter accountno: "))
balance=int(input("Enter balance: "))
typeoffaccount=input("Enter C for Current account, S for Saving
account: ")
if typeoffaccount.upper()=="S":
  Acc=sav_acct(name,accountno,balance)
elif typeoffaccount.upper()=="C":
  Acc=cur_acct(name,accountno,balance)
flag=True
while flag:
   print("1. Display balance")
   print("2. Deposit Money")
   print("3. Debit Money")
   if typeoffaccount.upper()=="S":
     print("4. Predict Balance")
   elif typeoffaccount.upper()=="C":
     print("4. Cheque Book")
   print("5. Exit")
   choice=int(input("Enter your choice: "))
   if choice==1:
     Acc.display()
   elif choice==2:
     Acc.deposit()
   elif choice==3:
     Acc.debit()
   elif choice==4:
```



```
            if typeoffaccount.upper()=="S":
                Acc.predictedBalance()
            elif typeoffaccount.upper()=="C":
                Acc.check_book()
        elif choice==5:
            flag=False

        option=input("Do you want to continue? (Y/N): ")
        if option.lower()=="n":
            flag=False
        else:
            flag=True
```

Output(s):



```
PS D:\testing> python -u "d:\testing\test.py"
Enter name: Arya
Enter accountno: 1234
Enter balance: 500
Enter C for Current account, S for Saving account: C
1. Display balance
2. Deposit Money
3. Debit Money
4. Cheque Book
5. Exit
Enter your choice: 4
Cheque book facility available
Do you want to continue? (Y/N): y
1. Display balance
2. Deposit Money
3. Debit Money
4. Cheque Book
5. Exit
Enter your choice: 1
Balance- Arya- 1234 - $500
Do you want to continue? (Y/N): y
1. Display balance
2. Deposit Money
3. Debit Money
4. Cheque Book
5. Exit
Enter your choice: 3
Enter the amount you want to debit- 300
Penalty imposed as balance below 500
Money Debited
Balance- Arya- 1234 - $196.0
Do you want to continue? (Y/N):
```

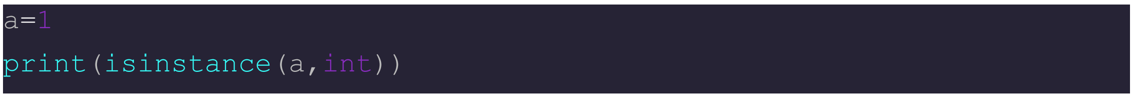
Conclusion:

We successfully completed the assignment with he knowledge of inheritance

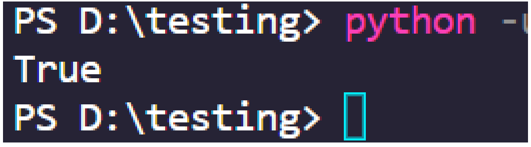
Post Lab Questions:

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

Python isinstance() function returns True if the object is specified types, and it will not match then return False.



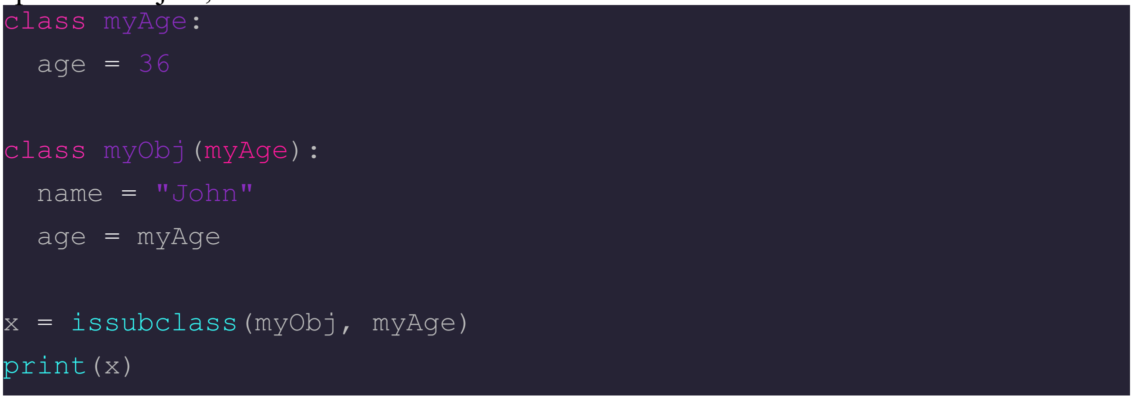
```
a=1
print(isinstance(a,int))
```



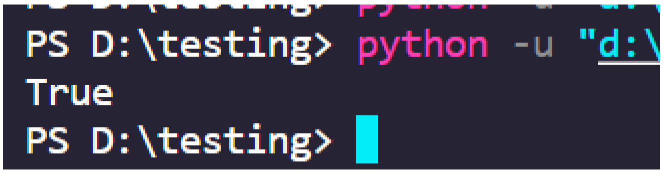
```
PS D:\testing> python -
True
PS D:\testing> 
```

as a is an integer it will be a instance of integer

The issubclass() function returns True if the specified object is a subclass of the specified object, otherwise False.



```
class myAge:
  age = 36

class myObj(myAge):
  name = "John"
  age = myAge

x = issubclass(myObj, myAge)
print(x)
```



```
PS D:\testing> python -u "d:\
True
PS D:\testing> 
```

as myObj inherits from myAge it is a subclass hence it returns true

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