Welcome to the Python_and_Deep_Learning_Course-CSEE5590 Lab-2 submission

By: Mani Sai Srinivas, Kandukuri

` Class-id: 20

Objective

This lab mainly consists of learning Python Objective-Oriented Programming.

First, second, fourth were done in "One" jupyter notebook(both .py and .ipnyb format was exported) and third was done in PyCharm.

All of the code is commented for understanding. Code can be found here

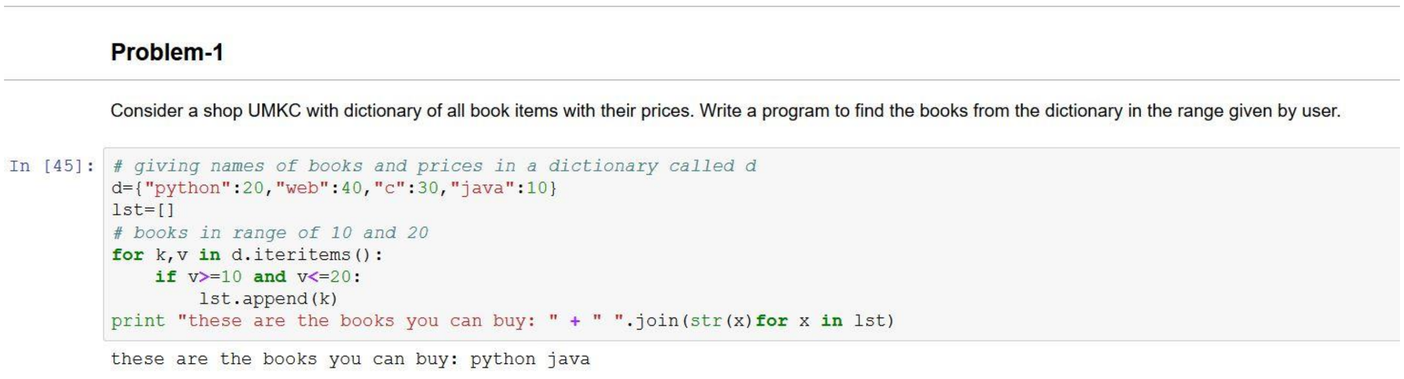
Configuration

- Python 2.7
- Jupyter Notebook
- PyCharm

Task-1

Consider a shop UMKC with dictionary of all book items with their prices. Write a program to find the books from the dictionary in the range given by use

Here is the screenshot of the code(both input and output):



Problem-1

Consider a shop UMKC with dictionary of all book items with their prices. Write a program to find the books from the dictionary in the range given by user.

In [45]:
```
# giving names of books and prices in a dictionary called d
d={"python":20,"web":40,"c":30,"java":10}
lst=[]
# books in range of 10 and 20
for k,v in d.iteritems():
    if v>=10 and v<=20:
        lst.append(k)
print "these are the books you can buy: " + " ".join(str(x)for x in lst)
```

```
these are the books you can buy: python java
```

Code Snippet:

```
# giving names of books and prices in a dictionary called d
d={"python":20,"web":40,"c":30,"java":10}
lst=[]
# books in range of 10 and 20
for k,v in d.iteritems():
    if v>=10 and v<=20:
        lst.append(k)
print "these are the books you can buy: " + " ".join(str(x)for x in lst)
```

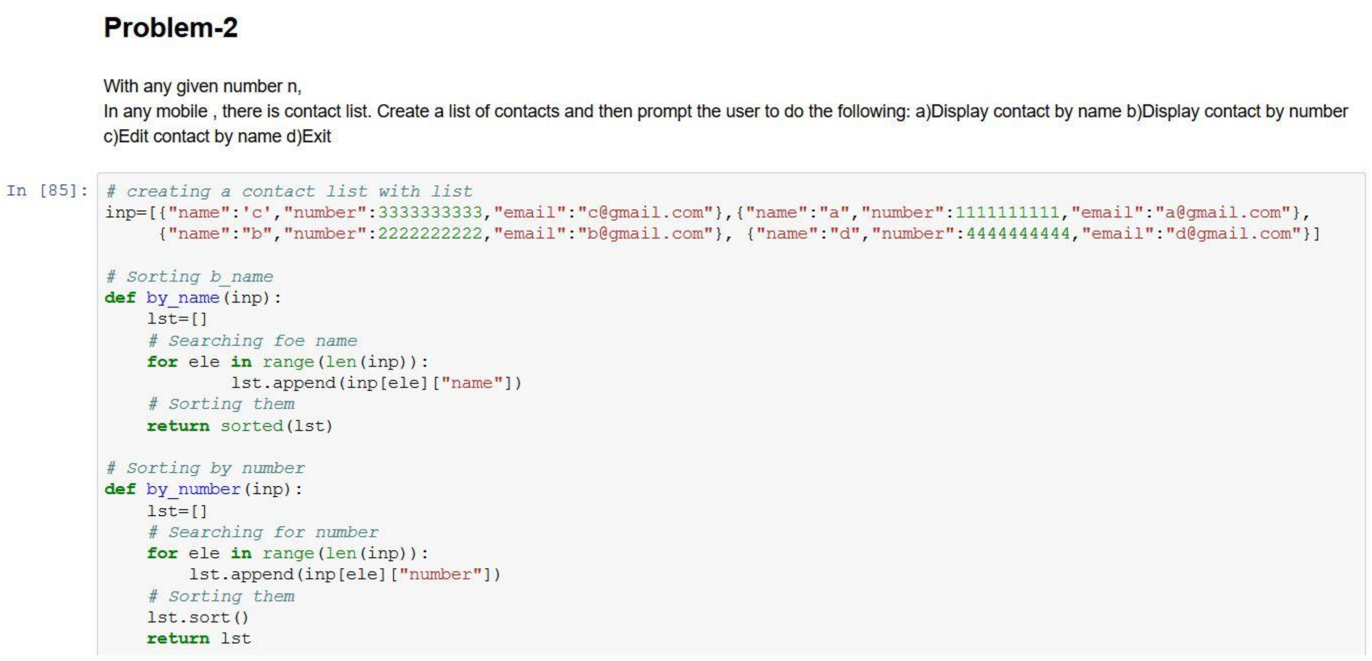
Output:

The output was java and python as they are in range

Task-2

With any given number n, In any mobile , there is contact list. Create a list of contacts and then prompt the user to do the following: a)Display contact by name b)Display contact by number c)Edit contact by name d)Exit

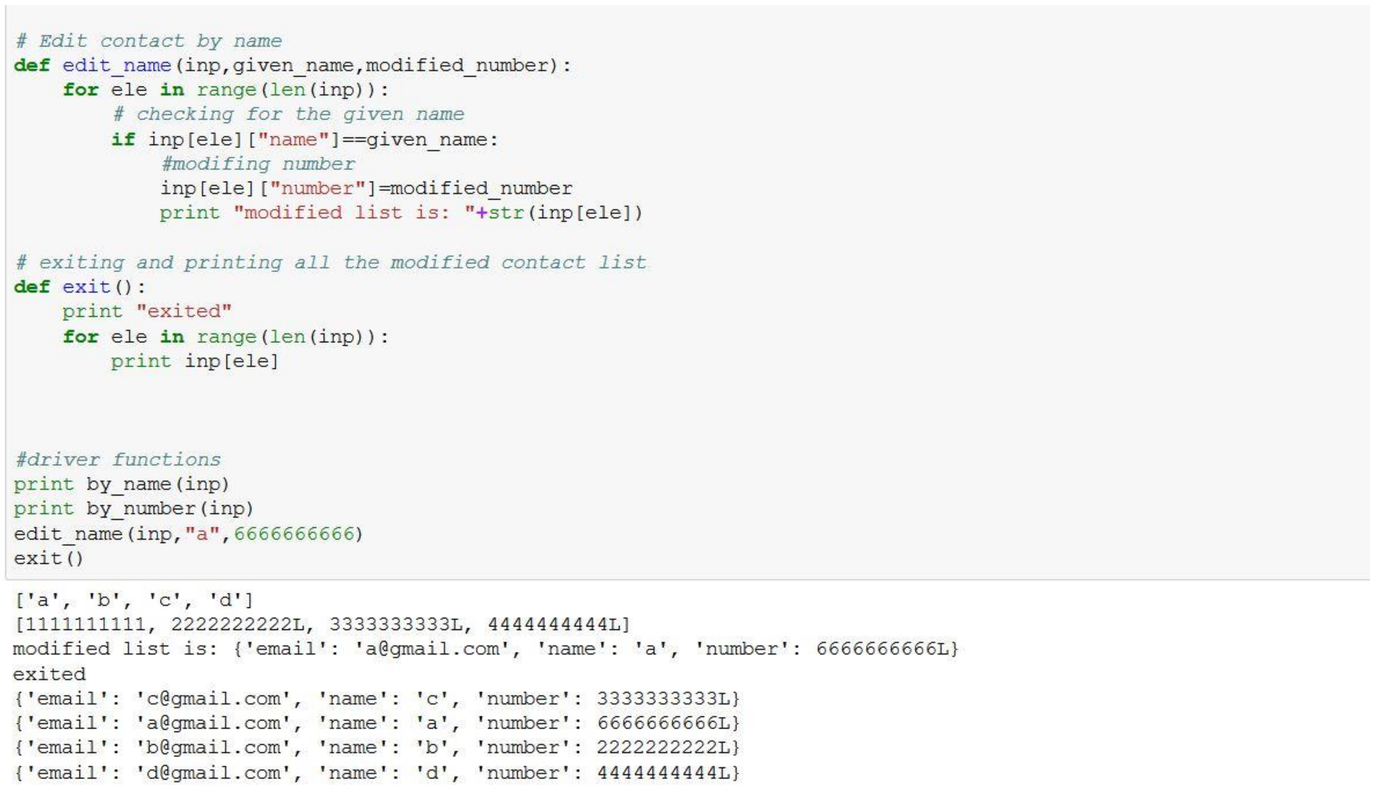
This is a screenshot of the code(both input and output):



Problem-2

With any given number n,
In any mobile , there is contact list. Create a list of contacts and then prompt the user to do the following: a)Display contact by name b)Display contact by number c)Edit contact by name d)Exit

In [85]:
```
# creating a contact list with list
inp=[{"name":'c',"number":3333333333,"email":"c@gmail.com"},{"name":"a","number":1111111111,"email":"a@gmail.com"},
     {"name":"b","number":2222222222,"email":"b@gmail.com"}, {"name":"d","number":4444444444,"email":"d@gmail.com"}]

# Sorting b_name
def by_name(inp):
    lst=[]
    # Searching foe name
    for ele in range(len(inp)):
          lst.append(inp[ele]["name"])
    # Sorting them
    return sorted(lst)

# Sorting by number
def by_number(inp):
    lst=[]
    # Searching for number
    for ele in range(len(inp)):
        lst.append(inp[ele]["number"])
    # Sorting them
    lst.sort()
    return lst
```



```
# Edit contact by name
def edit_name(inp,given_name,modified_number):
    for ele in range(len(inp)):
        # checking for the given name
        if inp[ele]["name"]==given_name:
            #modifing number
            inp[ele]["number"]=modified_number
            print "modified list is: "+str(inp[ele])

# exiting and printing all the modified contact list
def exit():
    print "exited"
    for ele in range(len(inp)):
        print inp[ele]



#driver functions
print by_name(inp)
print by_number(inp)
edit_name(inp,"a",6666666666)
exit()
```

```
['a', 'b', 'c', 'd']
[1111111111, 2222222222L, 3333333333L, 4444444444L]
modified list is: {'email': 'a@gmail.com', 'name': 'a', 'number': 6666666666L}
exited
{'email': 'c@gmail.com', 'name': 'c', 'number': 3333333333L}
{'email': 'a@gmail.com', 'name': 'a', 'number': 6666666666L}
{'email': 'b@gmail.com', 'name': 'b', 'number': 2222222222L}
{'email': 'd@gmail.com', 'name': 'd', 'number': 4444444444L}
```

Code Snippet:

```
inp = [{"name": 'c', "number": 3333333333, "email": "c@gmail.com"},
           {"name": "a", "number": 1111111111, "email": "a@gmail.com"},
           {"name": "b", "number": 2222222222, "email": "b@gmail.com"},
           {"name": "d", "number": 4444444444, "email": "d@gmail.com"}]


# Sorting b_name
def by_name(inp):
    lst = []
    # Searching foe name
    for ele in range(len(inp)):
        lst.append(inp[ele]["name"])
    # Sorting them
    return sorted(lst)


# Sorting by number
def by_number(inp):
    lst = []
    # Searching for number
    for ele in range(len(inp)):
        lst.append(inp[ele]["number"])
    # Sorting them
    lst.sort()
    return lst
```

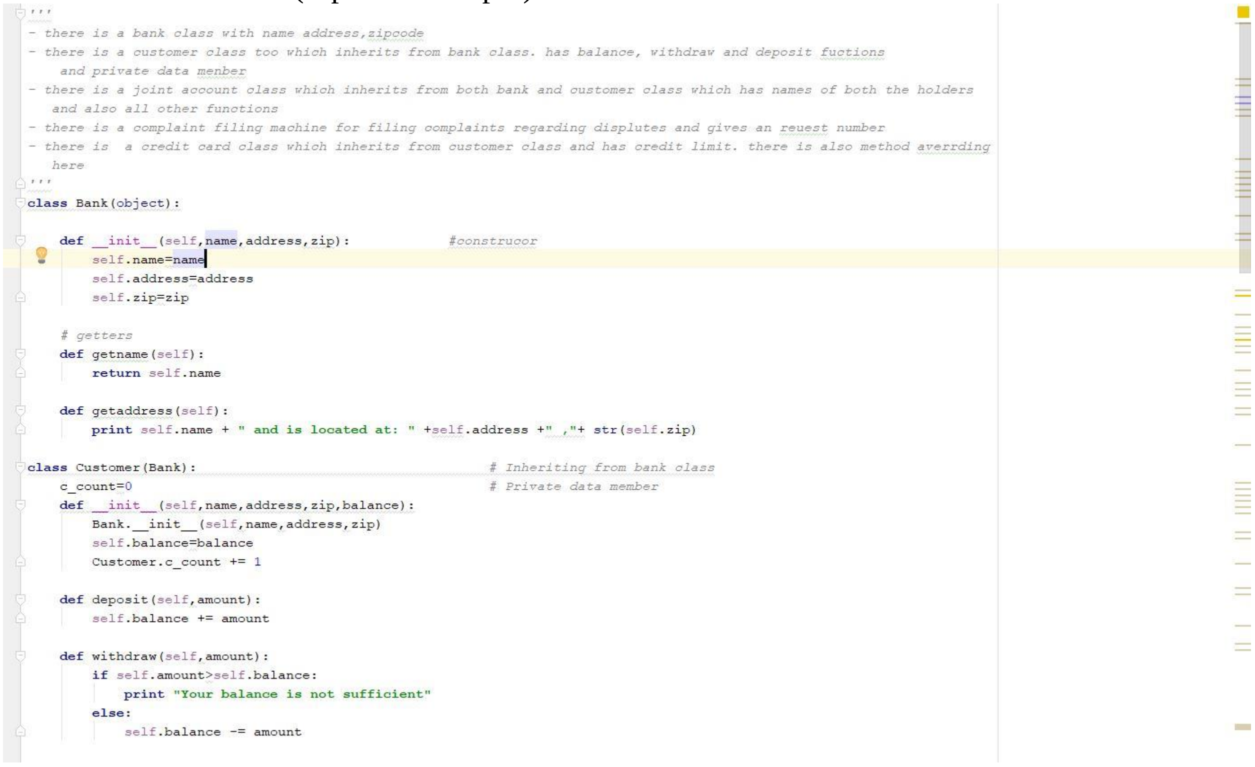
```
# Edit contact by name
def edit_name(inp, given_name, modified_number):
    for ele in range(len(inp)):
        # checking for the given name
        if inp[ele]["name"] == given_name:
            # modifing number
            inp[ele]["number"] = modified_number
            print "modified list is: " + str(inp[ele])


# exiting and printing all the modified contact list
def exit():
    print "exited"
    for ele in range(len(inp)):
        print inp[ele]


# driver functions
print by_name(inp)
print by_number(inp)
edit_name(inp, "a", 6666666666)
exit()
```

Task-3

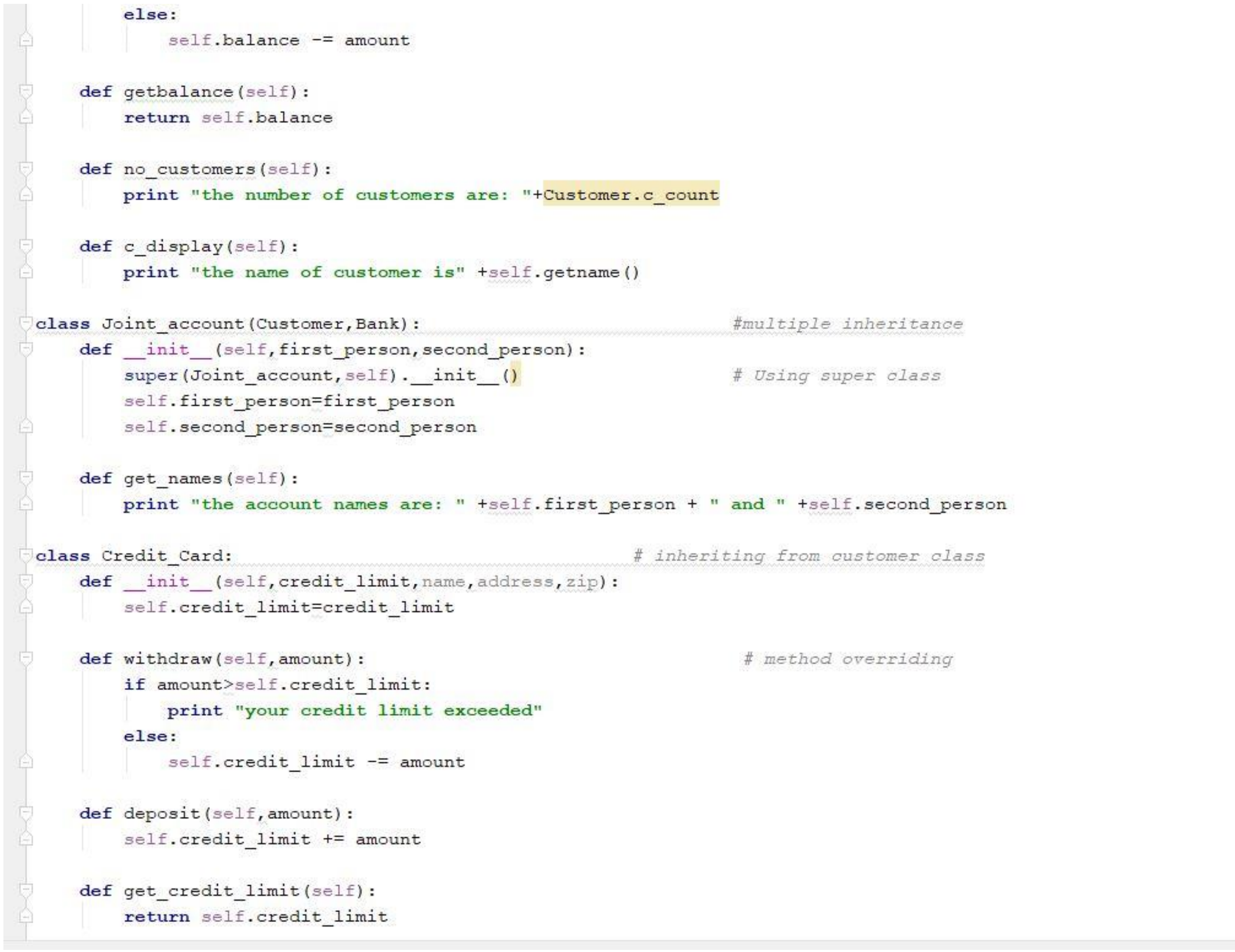
Write a python program to create any one of the following management systems. You can also pick one of your own.

I created a banking System

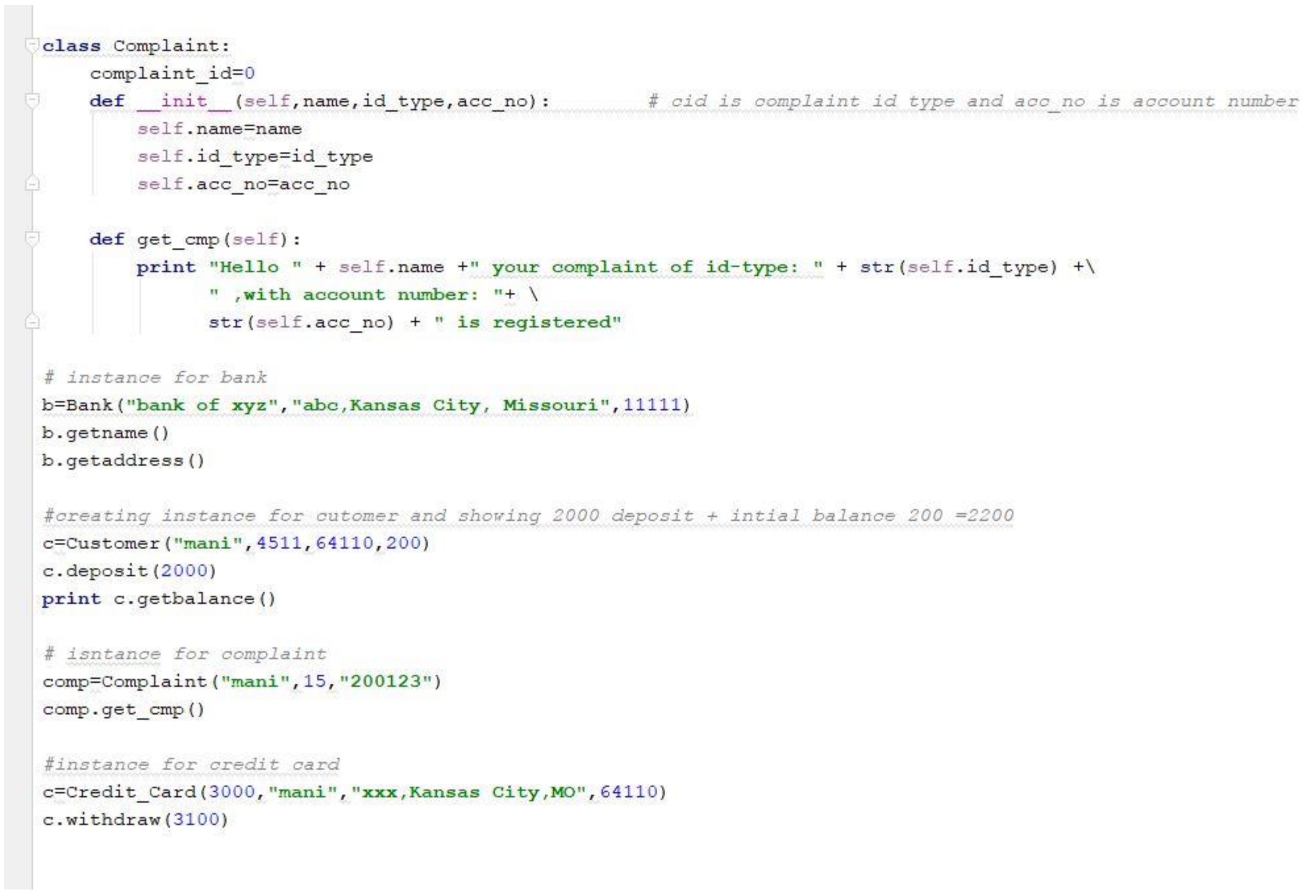
Screenshot of the code(Input and Output):



```
'''
- there is a bank class with name address,zipcode
- there is a customer class too which inherits from bank class. has balance, withdraw and deposit fuctions
    and private data menber
- there is a joint account class which inherits from both bank and customer class which has names of both the holders
   and also all other functions
- there is a complaint filing machine for filing complaints regarding displutes and gives an reuest number
- there is  a credit card class which inherits from customer class and has credit limit. there is also method averrding
   here
'''
class Bank(object):

    def __init__(self,name,address,zip):            #construcor
        self.name=name
        self.address=address
        self.zip=zip

    # getters
    def getname(self):
        return self.name

    def getaddress(self):
        print self.name + " and is located at: " +self.address +" ,"+ str(self.zip)

class Customer(Bank):                                    # Inheriting from bank class
    c_count=0                                            # Private data member
    def __init__(self,name,address,zip,balance):
        Bank.__init__(self,name,address,zip)
        self.balance=balance
        Customer.c_count += 1

    def deposit(self,amount):
        self.balance += amount

    def withdraw(self,amount):
        if self.amount>self.balance:
            print "Your balance is not sufficient"
        else:
            self.balance -= amount
```

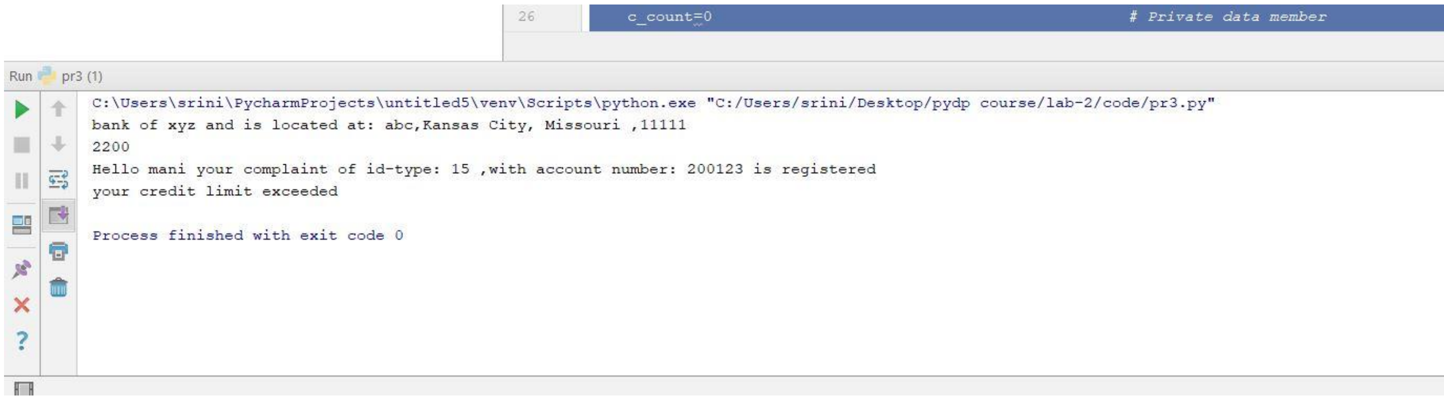


```
        else:
            self.balance -= amount

    def getbalance(self):
        return self.balance

    def no_customers(self):
        print "the number of customers are: "+Customer.c_count

    def c_display(self):
        print "the name of customer is" +self.getname()

class Joint_account(Customer,Bank):                            #multiple inheritance
    def __init__(self,first_person,second_person):
        super(Joint_account,self).__init__()                   # Using super class
        self.first_person=first_person
        self.second_person=second_person

    def get_names(self):
        print "the account names are: " +self.first_person + " and " +self.second_person

class Credit_Card:                                    # inheriting from customer class
    def __init__(self,credit_limit,name,address,zip):
        self.credit_limit=credit_limit

    def withdraw(self,amount):                                  # method overriding
        if amount>self.credit_limit:
            print "your credit limit exceeded"
        else:
            self.credit_limit -= amount

    def deposit(self,amount):
        self.credit_limit += amount

    def get_credit_limit(self):
        return self.credit_limit
```



```python
class Complaint:
    complaint_id=0
    def __init__(self,name,id_type,acc_no):         # cid is complaint id type and acc_no is account number
        self.name=name
        self.id_type=id_type
        self.acc_no=acc_no

    def get_cmp(self):
        print "Hello " + self.name +" your complaint of id-type: " + str(self.id_type) +\
              " ,with account number: "+ \
              str(self.acc_no) + " is registered"

# instance for bank
b=Bank("bank of xyz","abc,Kansas City, Missouri",11111)
b.getname()
b.getaddress()

#creating instance for cutomer and showing 2000 deposit + intial balance 200 =2200
c=Customer("mani",4511,64110,200)
c.deposit(2000)
print c.getbalance()

# isntance for complaint
comp=Complaint("mani",15,"200123")
comp.get_cmp()

#instance for credit card
c=Credit_Card(3000,"mani","xxx,Kansas City,MO",64110)
c.withdraw(3100)
```

Output:



```
        c_count=0                                          # Private data member
```

```
C:\Users\srini\PycharmProjects\untitled5\venv\Scripts\python.exe "C:/Users/srini/Desktop/pydp course/lab-2/code/pr3.py"
bank of xyz and is located at: abc,Kansas City, Missouri ,11111
2200
Hello mani your complaint of id-type: 15 ,with account number: 200123 is registered
your credit limit exceeded

Process finished with exit code 0
```

Code Snippet:

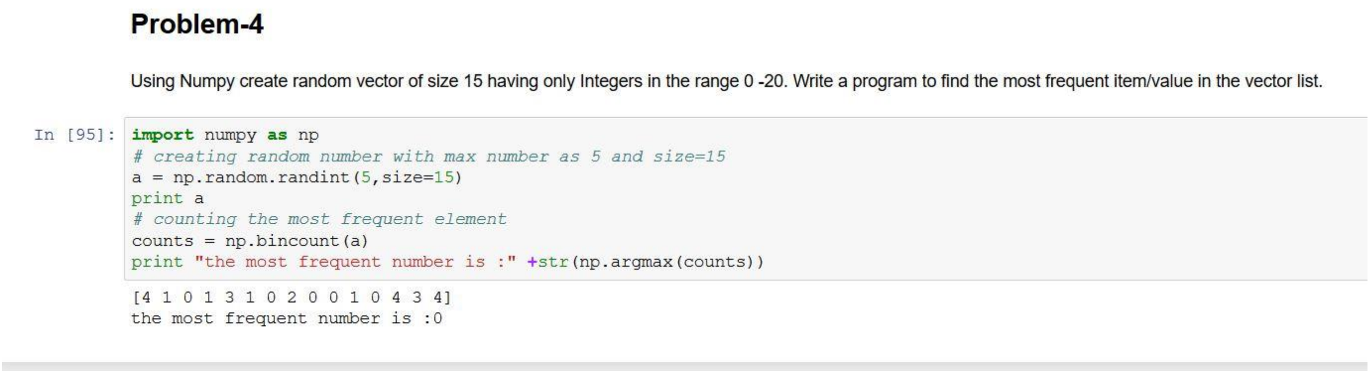
```
    '''
    - there is a bank class with name address,zipcode
    - there is a customer class too which inherits from bank class. has
balance, withdraw and deposit fuctions
        and private data menber
    - there is a joint account class which inherits from both bank and
customer class which has names of both the holders
```

```
       and also all other functions
    - there is a complaint filing machine for filing complaints regarding
displutes and gives an reuest number
    - there is  a credit card class which inherits from customer class and
has credit limit. there is also method averrding
       here
    '''
    class Bank(object):

        def __init__(self,name,address,zip):            #construcor
            self.name=name
            self.address=address
            self.zip=zip

        # getters
        def getname(self):
            return self.name

        def getaddress(self):
            print self.name + " and is located at: " +self.address +" ,"+
str(self.zip)

    class Customer(Bank):                                  # Inheriting
from bank class
        c_count=0                                          # Private data
member
        def __init__(self,name,address,zip,balance):
            Bank.__init__(self,name,address,zip)
            self.balance=balance
            Customer.c_count += 1

        def deposit(self,amount):
            self.balance += amount

        def withdraw(self,amount):
            if self.amount>self.balance:
                print "Your balance is not sufficient"
            else:
                self.balance -= amount

        def getbalance(self):
            return self.balance

        def no_customers(self):
            print "the number of customers are: "+Customer.c_count

        def c_display(self):
            print "the name of customer is" +self.getname()

    class Joint_account(Customer,Bank):                          #multiple
inheritance
        def __init__(self,first_person,second_person):
            super(Joint_account,self).__init__()                 # Using
super class
            self.first_person=first_person
            self.second_person=second_person
        def get_names(self):
            print "the account names are: " +self.first_person + " and "
+self.second_person

    class Credit_Card:                                    # inheriting from
customer class
        def __init__(self,credit_limit,name,address,zip):
            self.credit_limit=credit_limit

        def withdraw(self,amount):                                  # method
overriding
            if amount>self.credit_limit:
                print "your credit limit exceeded"
            else:
                self.credit_limit -= amount

        def deposit(self,amount):
            self.credit_limit += amount

        def get_credit_limit(self):
            return self.credit_limit

    class Complaint:
        complaint_id=0
        def __init__(self,name,id_type,acc_no):        # cid is complaint id
type and acc_no is account number
            self.name=name
            self.id_type=id_type
            self.acc_no=acc_no

        def get_cmp(self):
            print "Hello " + self.name +" your complaint of id-type: " +
str(self.id_type) +\
                  " ,with account number: "+ \
                  str(self.acc_no) + " is registered"

    # instance for bank
    b=Bank("bank of xyz","abc,Kansas City, Missouri",11111)
    b.getname()
    b.getaddress()

    #creating instance for cutomer and showing 2000 deposit + intial balance
200 =2200
    c=Customer("mani",4511,64110,200)
    c.deposit(2000)
    print c.getbalance()

    # isntance for complaint
    comp=Complaint("mani",15,"200123")
    comp.get_cmp()

    #instance for credit card
    c=Credit_Card(3000,"mani","xxx,Kansas City,MO",64110)
    c.withdraw(3100)
```

Task-4

Using Numpy create random vector of size 15 having only Integers in the range 0 -20. Write a program to find the most frequent item/value in the vector list.

Screenshot of the code(Input and Output):



Problem-4

Using Numpy create random vector of size 15 having only Integers in the range 0 -20. Write a program to find the most frequent item/value in the vector list.

```
In [95]: import numpy as np
         # creating random number with max number as 5 and size=15
         a = np.random.randint(5,size=15)
         print a
         # counting the most frequent element
         counts = np.bincount(a)
         print "the most frequent number is :" +str(np.argmax(counts))

         [4 1 0 1 3 1 0 2 0 0 1 0 4 3 4]
         the most frequent number is :0
```

Code Snippet:

```
import numpy as np
# creating random number with max number as 5 and size=15
a = np.random.randint(5,size=15)
print a
# counting the most frequent element
counts = np.bincount(a)
print "the most frequent number is :" +str(np.argmax(counts)
```

Limitations:

- For problem 3 we used many classes which is confusing. Instead we could have used 2 classes bank and customer class and implement the program.
- Using super class in multiple inheritance gives confusing regarding Python method resolution order (MRO). We can avoid
  this.

References: https://stackoverflow.com/