Name : Makwana Vishal Raghavbhai

Class : FYMCA (SAM-2) Div : B

Roll No: 113

Paper Number : 204

Subject : Python Programming Language

Assignment Number : 6

Dept. of Computer Science,
Veer Narmad South Gujarat University, Surat.

M.C.A. 2nd Semester (2020-2021)
Paper 204: Python Programming Language

Practical Sheet - 6

Q.1. Write a Menu Driven (Menu: PUSH, POP, PEEP, PRINT, EXIT) program in Python to implement stack operations on a stack of integers using a class consisting of attributes Stack (a List consisting of elements of the Stack) and TOP, and methods PUSH, POP, PEEP and PRINT.

```
class stack:
    def __init__(self, n):
        self.no = n
        self.Stack = []
        self.top = -1

    def push(self):
        if self.top == self.no - 1:
            print("Stack Overflow")
        else:
            n = int(input("enter an element : "))
            self.Stack.append(n)
            self.top += 1

    def pop(self):
        if self.top == -1:
            print("Stack Underflow....")
        else:
            self.Stack.pop()
            self.top -= 1
```

```
    def peep(self):
        print(self.top, "\t", self.Stack[-1])

    def disp(self):
        if self.top == -1:
            print("Stack Underflow")
        else:
            print("TOP \tELEMENT")
            for i in range(self.top, -1, -1):
                print(i, " \t", self.Stack[i])


n = int(input("Enter Size :: "))
stk = stack(n)
while (True):
    print(" 1: PUSH ")
    print(" 2: POP ")
    print(" 3: PEEP ")
    print(" 4: PRINT ")
    print(" 5: EXIT ")

    option = int(input("enter your choice :: "))

    if option == 1:
        stk.push()

    elif option == 2:
        stk.pop()

    elif option == 3:
        stk.peep()

    elif option == 4:
        stk.disp()

    elif option == 5:
        print("Exit")
        break
    else:
        print("Incorrect choice")
```

Q.2. Write a Menu Driven (Menu: INSERT, DELETE, PRINT, EXIT) program in Python to implement Simple Queue Operations on a simple queue of integers using a class consisting of attributes SQueue (a List consisting of elements of the Simple Queue), Front and Rear and methods INSERT, DELETE and PRINT.

```
class Queue:
    def __init__(self, no):
        self.no = no
        self.SQueue = []
        self.front = -1
        self.rear = -1

    def insert(self):
        if self.rear == self.no - 1:
            print("Queue is Full.....")
        else:
            if self.front == -1:
                self.front = 0
                self.rear = 0
            else:
                self.rear += 1
            n = int(input("enter an element :: "))
            self.SQueue.insert(self.rear, n)

    def delete(self):
        if self.front == -1 and self.front == no - 1:
            print("Queue is Empty.....")
        else:
            self.SQueue.pop(self.front)
            self.front += 1

    def disp(self):
        if self.front == -1 and self.front == no - 1:
            print("Queue is Empty.....")
        else:
            print("REAR \tELEMENT")
            for i in range(len(self.SQueue)):
                print(i, " \t", self.SQueue[i])


no = int(input("ENTER  Size :: "))
```

```
q = Queue(no)
while (True):
    print(" 1: INSERT ")
    print(" 2: DELETE ")
    print(" 3: PRINT ")
    print(" 4: EXIT ")
    option = int(input("enter your choice :: "))

    if option == 1:
        q.insert()

    elif option == 2:
        q.delete()

    elif option == 3:
        q.disp()

    elif option == 4:
        print("you are exit!!!!!")
        break
    else:
        print("Incorrect option")
```

Q.3. Write a Python program to do the following:

- **Define a class myDate consisting of attributes day, month and year and following methods:**
  - **addDays(myDate, int) where the 1st argument is a myDate class type and 2nd argument is an integer type with default value as 0. The method should add/subtract int days (depending on the integer i.e. add if positive and subtract if negative) to/from the myDate and return new date of myDate type.**

  - **formatDate(myDate, formatString) where the 1st argument is a myDate class type and 2nd argument is a format string of string type. The method will return the date in the given format. Consider only the following format strings in this program. 'dd-mm-yyyy', 'mm-dd-yyyy' and 'yyyy-mm-dd'.**

- **Implement the above**

```
import datetime

class MyDate:

    def __init__(self, day=0, month=0, year=0):
        self.day = day
        self.month = month
        self.year = year
        self.new_date = None

    def add_days(self, day=0):
        if day < 0:
            self.day -= day
        elif day > 0:
            self.day += day
        self.new_date = datetime.date(day=self.day,
                                      month=self.month,
```

```
                                        year=self.year)
        return self.new_date

    def format_date(self, date_format=None):
        try:
            if date_format.startswith("dd") and
                    date_format.endswith("yy"):
                return '{:%d-%m-%Y}'
                    .format(self.new_date)
            elif date_format.startswith("mm") and
                    date_format.endswith("yy"):
                return '{:%m-%d-%Y}'
                    .format(self.new_date)
            elif date_format.startswith("yy") and
                    date_format.
                            endswith("dd"):
                return '{:%Y-%m-%d}'
                    .format(self.new_date)
            else:
                return "Wrong format"
        except TypeError as e:
            return e


d = MyDate(5, 5, 2021)
print("enter a days ")
days = input(">>> ")
d.add_days(int(days))
print(d.format_date('yyyy-mm-dd'))
```

Q.4. Write a Python program to overload:

- **the '+' operator for the string as under:**
  The 2 strings should be merged in such a way that the result will contain characters one by one first from the 1st string and then from the 2nd string as shown in the example given below.
  Str1='VNSGU'
  Str2='SURAT'
  Then Str1+Str2 = 'VSNUSRGAUT' (i.e. here characters in Italics are from Str1 and
  rest from Str2)
- **the <, <=, >, >= and == operators for the strings as under:**
  Compare sum of ACII values of all the characters in both the strings and then compare the results. Return True if the sum for the 1st string is more than that for the 2nd string for the '>' operator and False otherwise. Similarly do for other operators.

```
class Operator_Overload:

    def __init__(self, str1):
        self.str1 = str1
        self.s1 = 0
        self.s2 = 0
    def __add__(self, other):
        result = ""
        i = 0
        while (i < len(self.str1)) or (i <
                len(other.str1)):
            if (i < len(self.str1)):
                result += self.str1[i]
            if (i < len(other.str1)):
                result += other.str1[i]
            i += 1
        return result
    def __lt__(self, other):
        i = 0
```

```
        while (i < len(self.str1)):
            self.s1 += ord(self.str1[i])
            i += 1
            i = 0
        while (i < len(other.str1)):
            self.s2 += ord(other.str1[i])
            i += 1
        if self.s1 < self.s2:
            return True
        else:
            return False
    def __le__(self, other):
        i = 0
        while (i < len(self.str1)):
            self.s1 += ord(self.str1[i])
            i += 1
            i = 0
        while (i < len(other.str1)):
            self.s2 += ord(other.str1[i])
            i += 1
        if self.s1 <= self.s2:
            return True
        else:
            return False
    def __eq__(self, other):
        i = 0
        while (i < len(self.str1)):
            self.s1 += ord(self.str1[i])
            i += 1
            i = 0
        while (i < len(other.str1)):
            self.s2 += ord(other.str1[i])
            i += 1
        if self.s1 == self.s2:
            return True
        else:
            return False
    def __gt__(self, other):
        i = 0
        while (i < len(self.str1)):
            self.s1 += ord(self.str1[i])
            i += 1
            i = 0
        while (i < len(other.str1)):
```

```
            self.s2 += ord(other.str1[i])
            i += 1
        if self.s1 > self.s2:
            return True
        else:
            return False
    def __ge__(self, other):
        i = 0
        while (i < len(self.str1)):
            self.s1 += ord(self.str1[i])
            i += 1
            i = 0
        while (i < len(other.str1)):
            self.s2 += ord(other.str1[i])
            i += 1
        if self.s1 >= self.s2:
            return True
        else:
            return False

obj1 = Operator_Overload(input("enter 1st string
                                :"))
obj2 = Operator_Overload(input("enter 2nd string
                                :"))
print(obj1 + obj2)
print(obj1 < obj2)
print(obj1 > obj2)
print(obj1 <= obj2)
print(obj1 >= obj2)
print(obj1 == obj2)
```

Q.5. Write a Python program for the following:

- **Define a class accountHolder consisting of attributes accNo, accName, accEmail. It consists a method dispDetails to display the details in appropriate format.**
- **Inherit two classes viz. depositAccount (accountBalance) and loanAccount(loanAmount, EMI, loanBalance). The depositAccount contains methods debitAmt(amt)-which debits amt amount from the accountBalance, creditAmt(amt)-**
  which credits the amt amount to the accountBalance and dispTrans()-which displays the whole transaction in appropriate format showing initial balance, debit/credit amount and final amount.

```
class depositeAccount:
    accountBalance = 400

    def debitAmt(self, amt):

        if amt < self.accountBalance:
            self.accountBalance = self.accountBalance - amt
        else:
            print("BALANCE IS NOT ENOUGH!")

    def creditAmt(self, amt):

        self.accountBalance = self.accountBalance + amt

    def dispTrans(self):
        print("account balance:",
                    depositeAccount.accountBalance)

        if depositeAccount.accountBalance >
           self.accountBalance:

            print("debit amount: ",
                  depositeAccount.accountBalance -
                  self.accountBalance)

        elif depositeAccount.accountBalance <
             self.accountBalance:
            print("credit amount ", self.accountBalance -
                  depositeAccount.accountBalance)
```

```
        print("final balance : ", self.accountBalance)


class loanAmount:
    loanAmount = 10000
    EMI = 100
    loanBalance = 7500


class accountHolder(depositeAccount, loanAmount):
    accNo = 110000857
    accName = "vishal makwana"
    accEmail = "makwanavishal8306@gmail.com"

    def dispDetail(self):
        print("ACCOUNT NUMBER ACCOUNTNAME ACCOUNTEMAIL")
        print(self.accNo, " " + self.accName + " " +
              self.accEmail)

obj = accountHolder()
obj.dispDetail()

print("1 -  DEBIT BALANCE")
print("2 -  CREDIT BALANCE")
ch = input("ENTER YOUR CHOICE : ")

if ch == '1' or ch == 'debit' or ch == 'DEBIT':
    n = int(input("ENTER AMOUNT YOU WANT TO DEBIT : "))
    obj.debitAmt(n)

if ch == '2' or ch == 'credit' or ch == 'CREDIT':
    n = int(input("ENTER AMOUNT YOU WANT TO CREDIT : "))
    obj.creditAmt(n)
obj.dispTrans()
```

Q.6. Write a Python program to demonstrate multi-level, multiple inheritance and MRO.

```
class student:
    def RollNo(self, RollNo):
        self.RollNo = RollNo


class test(student):
    def marks(self, mark1, mark2):
        self.mark1 = mark1
        self.mark2 = mark2


class sports(student):
    def score(self, scr):
        self.scr = scr


class result(test, sports):
    def display(self):
        self.RollNo(3)
        self.marks(20, 30)
        self.score(50)
        total = self.mark1 + self.mark2 + self.scr
        percentage = total / 3
        print("student information".center(200, "-"))
        print("RollNo :: ", self.RollNo, "Total::",
              total, "Percentage:: ", percentage)


obj = result()
obj.display()

print("Multiple Inheritance MRO".center(200, "-"))
print(result.__mro__)
print("Multi-Level Inheritance MRO".center(200, "-"))
print(result.__mro__)
```

Q.7. Write a Python program to demonstrate polymorphism using appropriate example.

```
class poly:

    def add(self, a=5, b=None, c=None):
        if a != None and b != None and c != None:
            d = a + b + c
        elif a != None and b != None:
            d = a + b
        else:
            d = a
        print(d)

    # constructor
    def __init__(self):
        self.a = 20

    ## operator overloading
    def __add__(self, res):
        obj = poly()
        obj.a = self.a + res.a
        return obj

    def disp(self):
        print(self.a)


class subPoly(poly):
    # method overriding
    def add(self, a=None, b=None, c=None):
        if a != None and b != None and c != None:
            d = a + b + c
        elif a != None and b != None:
            d = a + b
        else:
            d = a
        print(d)


## method overridding subpoly's add method call
obj = subPoly()
obj.add("vishal", "R", "makwana")
```

```
obj.add("vishal", "makwana")
obj.add("vishal")

## operator overloading
obj1 = poly()  # object of poly clASS init method
call
# obj1.add(3,5,7)
# obj.add(5,5)
# obj.add(5)

obj2 = poly()  # object of poly clASS init method
call
obj3 = poly()

## operator overloading
obj3 = obj1 + obj2
obj3.disp()
```

****