Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

Institute of Computer Technology
B. Tech Computer Science and Engineering

Sub: (2CSE403) FUNCTIONAL PROGRAMMING

Practical 8

1. A bank application corresponding to the customer of ABC bank is being developed. It takes into consideration Name of customer, its account type (saving or current), balance corresponding to the account. Account type is by default fixed from the day when user creates and account in a bank. Withdrawal & deposit are other operations which any customer would do. Also implement function called setbalance() & getbalance(). Ensure that minimum amount to be maintained is 1000/- in both savings and current. An error should be raised if such scenario occurs.

Code :

```
from YSL_io import *


class BnkAc:
    def __init__(self, name, ac, blnc):
        self.name = name
        self.ac = ac
        self.blnc = blnc
        self.min_blnc = 1000

        if self.blnc < self.min_blnc:
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

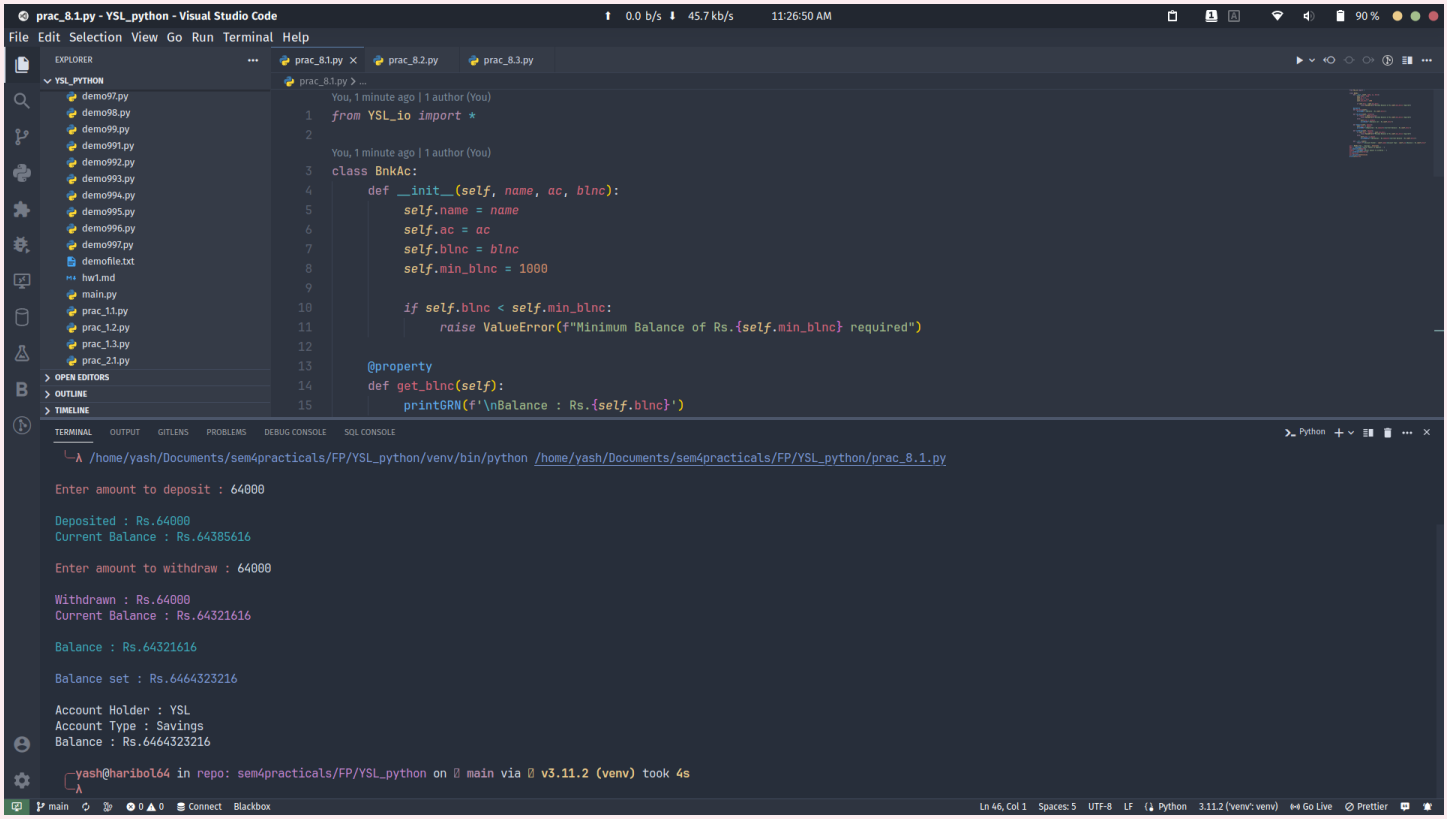
```
            raise ValueError(f"Minimum Balance of Rs.{self.min_blnc}
required")

    @property
    def get_blnc(self):
        printGRN(f'\nBalance : Rs.{self.blnc}')

    def set_blnc(self, setblnc):
        if setblnc < self.min_blnc:
            raise ValueError(f"Minimum Balance of Rs.{self.min_blnc}
required")
        else:
            self.blnc = setblnc
            printBLU(f"\nBalance set : Rs.{self.blnc}")

    def deposit(self, amount):
        self.blnc += amount
        printGRN(f'\nDeposited : Rs.{amount}\nCurrent Balance :
Rs.{self.blnc}')

    def withdraw(self, amount):
        if self.blnc - amount < self.min_blnc:
            raise ValueError(f"Minimum Balance of Rs.{self.min_blnc}
required")
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

```
        else:
            self.blnc -= amount
            printMGNTA(f"\nWithdrawn : Rs.{amount}\nCurrent Balance :
Rs.{self.blnc}")

    def __str__(self):
        return f"\nAccount Holder : {self.name}\nAccount Type :
{self.ac}\nBalance : Rs.{self.blnc}"

ysl = BnkAc('YSL', 'Savings', 64321616)
dpst = inputRED('\nEnter amount to deposit : ')
ysl.deposit(int(dpst))
wthdrw = inputRED('\nEnter amount to withdraw : ')
ysl.withdraw(int(wthdrw))
ysl.get_blnc
ysl.set_blnc(6464323216)
print(str(ysl))
```

Output :

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7



2. Design a class named StopWatch. The class contains:

   ■ The private data fields startTime and endTime with get methods.
   ■ A constructor that initializes startTime with the current time.
   ■ A method named start() that resets the startTime to the current time.
   ■ A method named stop() that sets the endTime to the current time.
   ■ A method named getElapsedTime() that returns the elapsed time for the stop watch in milliseconds.

   Write a test program that measures the execution time of adding numbers from 1 to 1,000,000.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

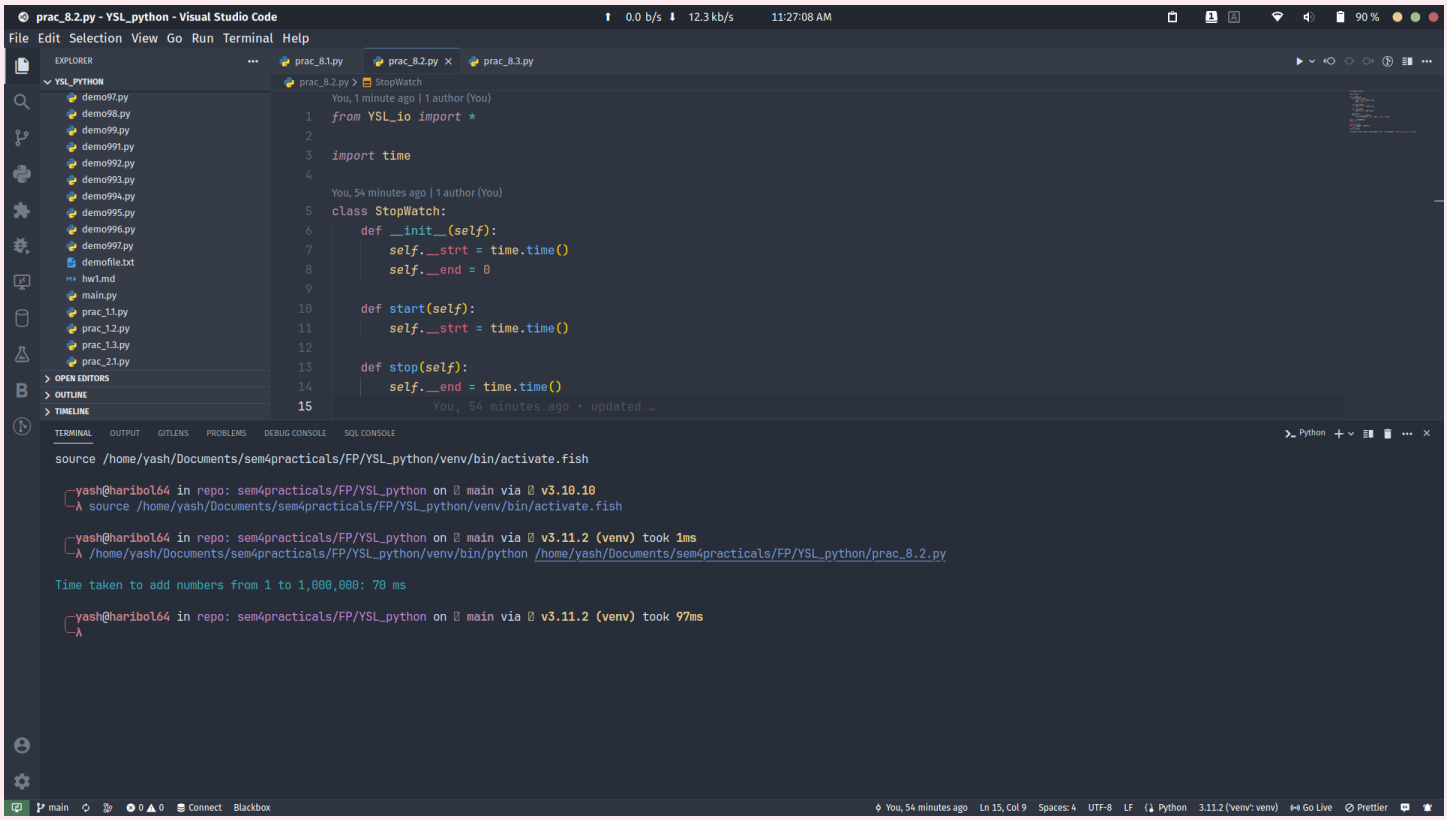
Code :

```
from YSL_io import *

import time

class StopWatch:
    def __init__(self):
        self.__strt = time.time()
        self.__end = 0

    def start(self):
        self.__strt = time.time()

    def stop(self):
        self.__end = time.time()

    @property
    def calculate_time(self):
        return int((self.__end - self.__strt) * 1000)

stpwtch = StopWatch()
total = 0

stpwtch.start()
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

```
for i in range(1, 1000001):
    total += i
stpwtch.stop()

printGRN(f"\nTime taken to add numbers from 1 to 1,000,000:
{stpwtch.calculate_time} ms")
```

Output :



Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

3. A small module for manipulation of complex numbers is being developed for ease of research related work. Implement add() for addition, mul() for multiplication , sub() for subtraction of two complex numbers. By default, a complex number will be assigned an imaginary value of 3. Also ensure that mul() cannot be ever 0 ; so program should raise an error.

Code :

```
from YSL_io import *


class Complex:
    def __init__(self, real, img=3):
        self.real = real
        self.img = img

    def __add__(self, other):
        print(f"Addition of {self} and {other} : ", end=' ')
        printORNG(f'{self.real + other.real} + {self.img + other.img}i')

    def __sub__(self, other):
        print(f"Subtraction of {self} and {other} : ", end=' ')
        printORNG(f'{self.real - other.real} + {self.img - other.img}i')

    def __mul__(self, other):
        result_real = self.real * other.real - self.img * other.img
        result_img = self.real * other.img + self.img * other.real
        if result_real == 0 and result_img == 0:
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

```
            raise ValueError("Multiplication result cannot be zero")
        print(f"Multiplication of {self} and {other} : ", end=' ')
        printORNG(f'{result_real} + {result_img}i')

    def __str__(self):
        return f"{self.real} + {self.img}i"

r1 = int(inputGRN('\nEnter the real coefficient of a complex number 1 :
'))
i1 = int(inputGRN('Enter the imaginary coefficient of a complex number 1 :
'))
c1 = Complex(r1, i1)

r2 = int(inputGRN('Enter the real coefficient of a complex number 2 : '))
i2 = int(inputGRN('Enter the imaginary coefficient of a complex number 2 :
'))
c2 = Complex(r2, i2)

print('\n')
c1 + c2
c1 - c2
c1 * c2
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 41
FP Practical 7

Output :

prac_8.3.py - YSL_python - Visual Studio Code

```python
from YSL_io import *

class Complex:
    def __init__(self, real, img=3):
        self.real = real
        self.img = img

    def __add__(self, other):
        print(f"Addition of {self} and {other} : ", end=' ')
        printORNG(f'{self.real + other.real} + {self.img + other.img}i')

    def __sub__(self, other):
        print(f"Subtraction of {self} and {other} : ", end=' ')
        printORNG(f'{self.real - other.real} + {self.img - other.img}i')
```

```
source /home/yash/Documents/sem4practicals/FP/YSL_python/venv/bin/activate.fish

yash@haribol64 in repo: sem4practicals/FP/YSL_python on main [!] via v3.10.10
λ source /home/yash/Documents/sem4practicals/FP/YSL_python/venv/bin/activate.fish

yash@haribol64 in repo: sem4practicals/FP/YSL_python on main [!] via v3.11.2 (venv) took 4ms
λ /home/yash/Documents/sem4practicals/FP/YSL_python/venv/bin/python /home/yash/Documents/sem4practicals/FP/YSL_python/prac_8.3.py

Enter the real coefficient of a complex number 1 : 9
Enter the imaginary coefficient of a complex number 1 : 5
Enter the real coefficient of a complex number 2 : 6
Enter the imaginary coefficient of a complex number 2 : 8


Addition of 9 + 5i and 6 + 8i :  15 + 13i
Subtraction of 9 + 5i and 6 + 8i :  3 + -3i
Multiplication of 9 + 5i and 6 + 8i :  14 + 102i

yash@haribol64 in repo: sem4practicals/FP/YSL_python on main [!] via v3.11.2 (venv) took 4s
λ
```