CSCI 1411: Fundamentals of Computing
Lab 13
Due Date: 8:30 AM November 17, 2020

Name: Kerry Gip

Goals:
- Classes
- Objects
- PyDoc

Development Environment: IDLE

Deliverables:
1. This lab handout with 4 screen shots (2 for part I and 2 for part II).
2. Your Python code for Part II of this lab (BankAccount.py and Register.py).

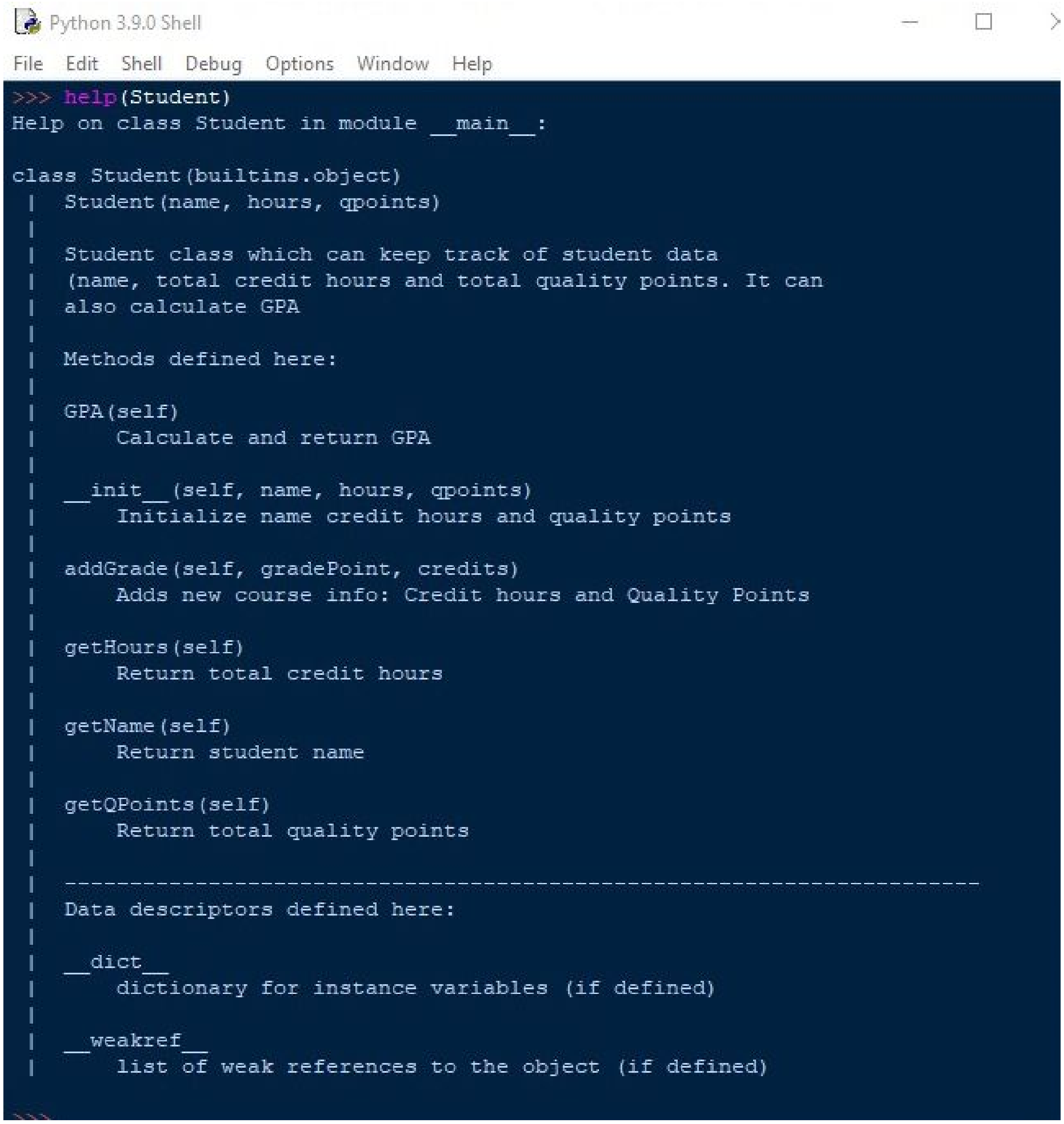
How to take a **screen shot**:
- For a Windows 10: Use Snipping Tool to copy and CTRL + V to paste screen shot.
- For Mac: Shift + Command + 4 to copy and CTRL + V to paste screen shot.

Part I – Skill Practice (10 Points)
- Start IDLE
- Create a new file
- You will create two .py files: Student.py (will contain code for Student class) and gpa_calculator.py (will contain code for main function)
- Type the following code in the file. Do not cut and paste. You will learn more by typing it in.
- Make sure that you read all comments to understand the code
- Remember to update the first line with your own name and the date of the lab.
- We are using PyDoc style comments in Student.py file (Student class)

```
# Student.py
# Name:
# Date:

class Student:
    """Student class which can keep track of student data
    (name, total credit hours, and total quality points. It
    can also calculate gpa"""


    def __init__(self, name, hours, qpoints):
        """Initialize name, credit hours, and quality points"""
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)

    def getName(self):
        """Return student's name"""
        return self.name

    def getHours(self):
        """Return total credit hours"""
        return self.hours

    def getQPoints(self):
        """Return total quality points"""
        return self.qpoints

    def gpa(self):
        """Calculate and return gpa"""
        return self.qpoints/self.hours

    def add_grade (self, grade_point, credits):
        """Add a new course information (Credit hours and Quality
        Points)"""
        self.hours = self.hours + credits
        self.qpoints = self.qpoints + grade_point
```

```
# gpa_calculator.py
# Name:
# Date:

#Import Student class
from Student import *

def main():
    name = input ('Enter your name: ')

    # Create a student object with 0 credit and 0 quality points
    s1 = Student (name, 0,0)

    # Ask for number of courses and grades
    count = int (input ('Enter number of grades: '))

    # Ask for credit hours and total quality point for each course
    # and add the course using add_grade method in student class.
    for i in range (count):
        credit = float (input('Enter credit hours for course ' + str(i+1) + ': '))
        gp = float (input ('Enter grade points for course ' + str(i+1) + ': '))
        s1.add_grade (gp, credit)

    # Display information include student name, total credit hours, total
    # quality points and gpa
    print ('Student name:', name)
    print ('Total Credit Hours {0:.2f}'.format(s1.getHours()))
    print ('Total Quality Points {0:.2f}'.format(s1.getQPoints()))
    print ('GPA {0:.2f}'.format(s1.gpa()))
```

- Save the files `Student.py` and `gpa_calculator.py`
- Click Run -> Run Module
- Type `help(Student)` in shell and it will display PyDoc comments. **Take a screen shot and paste it below**
- Type `main()` in shell to run your program
- If there are any syntax errors then fix those errors and run your program again.



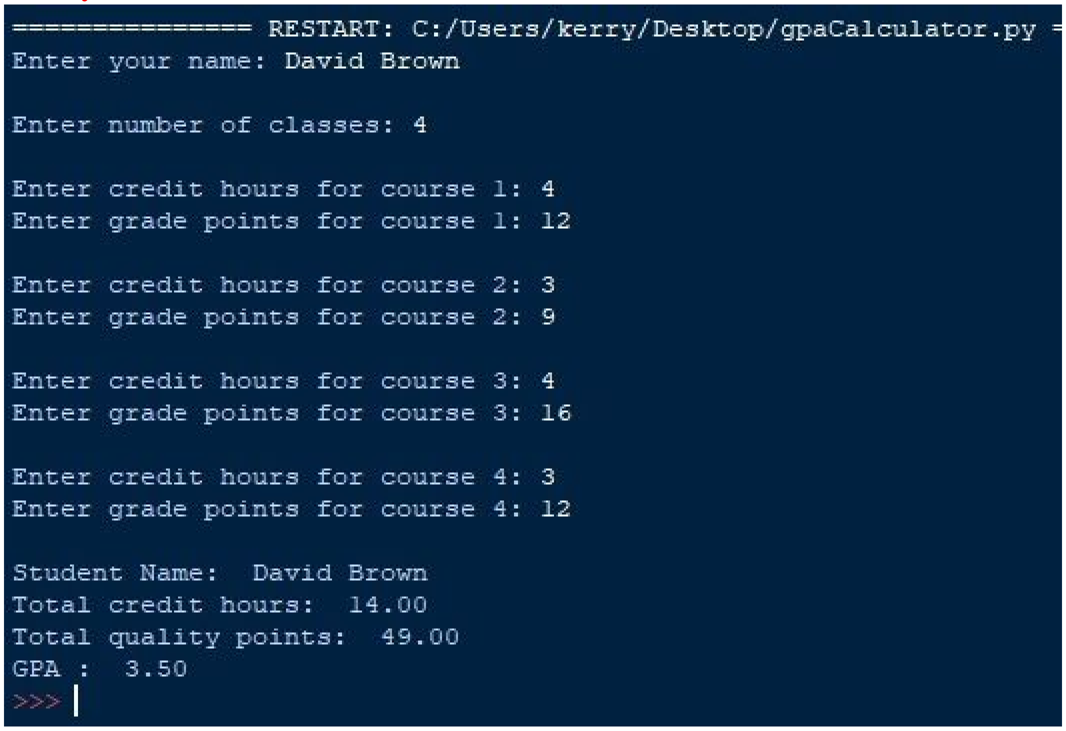
```
>>> help(Student)
Help on class Student in module __main__:

class Student(builtins.object)
 |  Student(name, hours, qpoints)
 |
 |  Student class which can keep track of student data
 |  (name, total credit hours and total quality points. It can
 |  also calculate GPA
 |
 |  Methods defined here:
 |
 |  GPA(self)
 |      Calculate and return GPA
 |
 |  __init__(self, name, hours, qpoints)
 |      Initialize name credit hours and quality points
 |
 |  addGrade(self, gradePoint, credits)
 |      Adds new course info: Credit hours and Quality Points
 |
 |  getHours(self)
 |      Return total credit hours
 |
 |  getName(self)
 |      Return student name
 |
 |  getQPoints(self)
 |      Return total quality points
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

>>>
```

- Use the following input to test your program:

```
Enter your name: David Brown
Enter number of grades: 4
Enter credit hours for course 1: 4
Enter grade points for course 1: 12
Enter credit hours for course 2: 3
Enter grade points for course 2: 9
Enter credit hours for course 3: 4
Enter grade points for course 3: 16
Enter credit hours for course 4: 3
Enter grade points for course 4: 12
```

- You will get following output:
```
Student name: David Brown
Total Credit Hours 14.00
Total Quality Points 49.00
GPA 3.50
```
- If you get the correct result then your program is working as expected.
- **Once you are satisfied with your results take a screen shot and paste them below.**

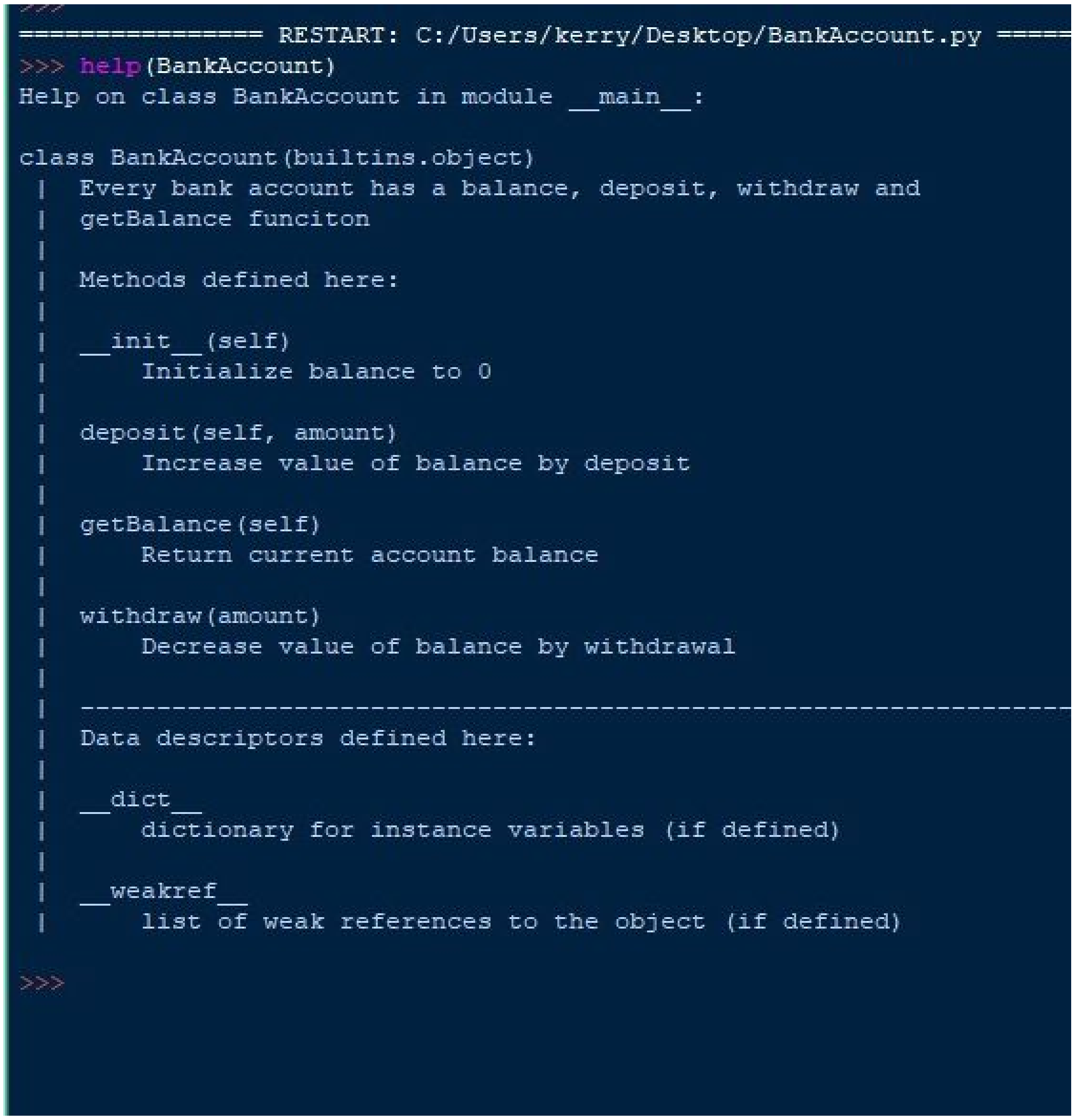
Paste your screen shots below this line



```
=============== RESTART: C:/Users/kerry/Desktop/gpaCalculator.py =
Enter your name: David Brown

Enter number of classes: 4

Enter credit hours for course 1: 4
Enter grade points for course 1: 12

Enter credit hours for course 2: 3
Enter grade points for course 2: 9

Enter credit hours for course 3: 4
Enter grade points for course 3: 16

Enter credit hours for course 4: 3
Enter grade points for course 4: 12

Student Name:  David Brown
Total credit hours:  14.00
Total quality points:  49.00
GPA :  3.50
>>> 
```

Part II – Bank Account Transactions (15 Points)

- Implement a class named `BankAccount`. Every bank account has a starting balance of $0.00. The class should implement methods to accept deposits and make withdrawals.
  - `__init__(self)` **:** Sets the balance to 0.
  - `deposit(self, amount)` **:** Deposits money. Return `True` if transaction is successful. Return `False` if amount is less than 0 and ignore the transaction.
  - `withdraw(self, amount)` **:** Withdraws money. Return `True` if transaction is successful. Return `False` if amount more than the balance and ignore the transaction.
  - `getBalance(self)` **:** Returns the amount of money in the account.
- Include PyDoc comments for your class and methods.
- Write a program with main function which will perform the following tasks:
  - Create a `BankAccount` object
  - Ask user for the number of transactions.
  - For each transaction ask for type of transaction and amount of transaction.
  - If type is deposit then use `deposit` method to complete the transaction. If return value from the `deposit` method is `False` then display an error message.
  - If type of the transaction is withdraw then use the `withdraw` method to complete the transaction. If return value from the `withdraw` method is `False` then display an error message.
  - After the loop display the number of transactions completed and account balance. If any transaction is rejected then it will not be included in the count of completed transactions.
- Save the files `BankAccount.py` and `Register.py`
- Click Run -> Run Module
- Type `help(BankAccount)` in shell and it will display PyDoc comments. **Take a screen shot and paste it below**



```
>>>
================ RESTART: C:/Users/kerry/Desktop/BankAccount.py =====
>>> help(BankAccount)
Help on class BankAccount in module __main__:

class BankAccount(builtins.object)
 |  Every bank account has a balance, deposit, withdraw and
 |  getBalance funciton
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      Initialize balance to 0
 |
 |  deposit(self, amount)
 |      Increase value of balance by deposit
 |
 |  getBalance(self)
 |      Return current account balance
 |
 |  withdraw(amount)
 |      Decrease value of balance by withdrawal
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

>>>
```

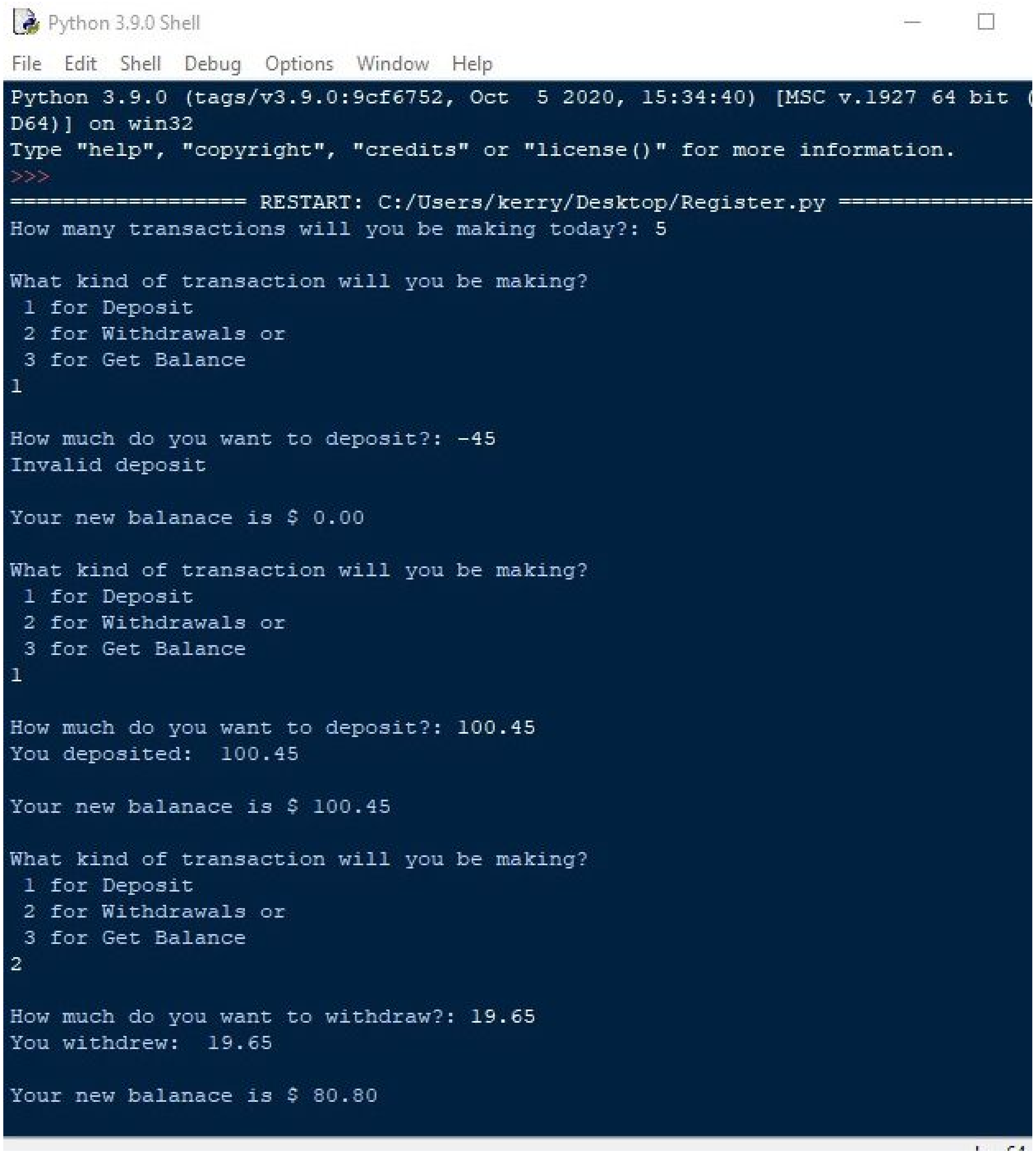
- Type `main()` in shell to run your program
- If there are any syntax errors then fix those errors and run your program again.

- Sample I/O is as follows:

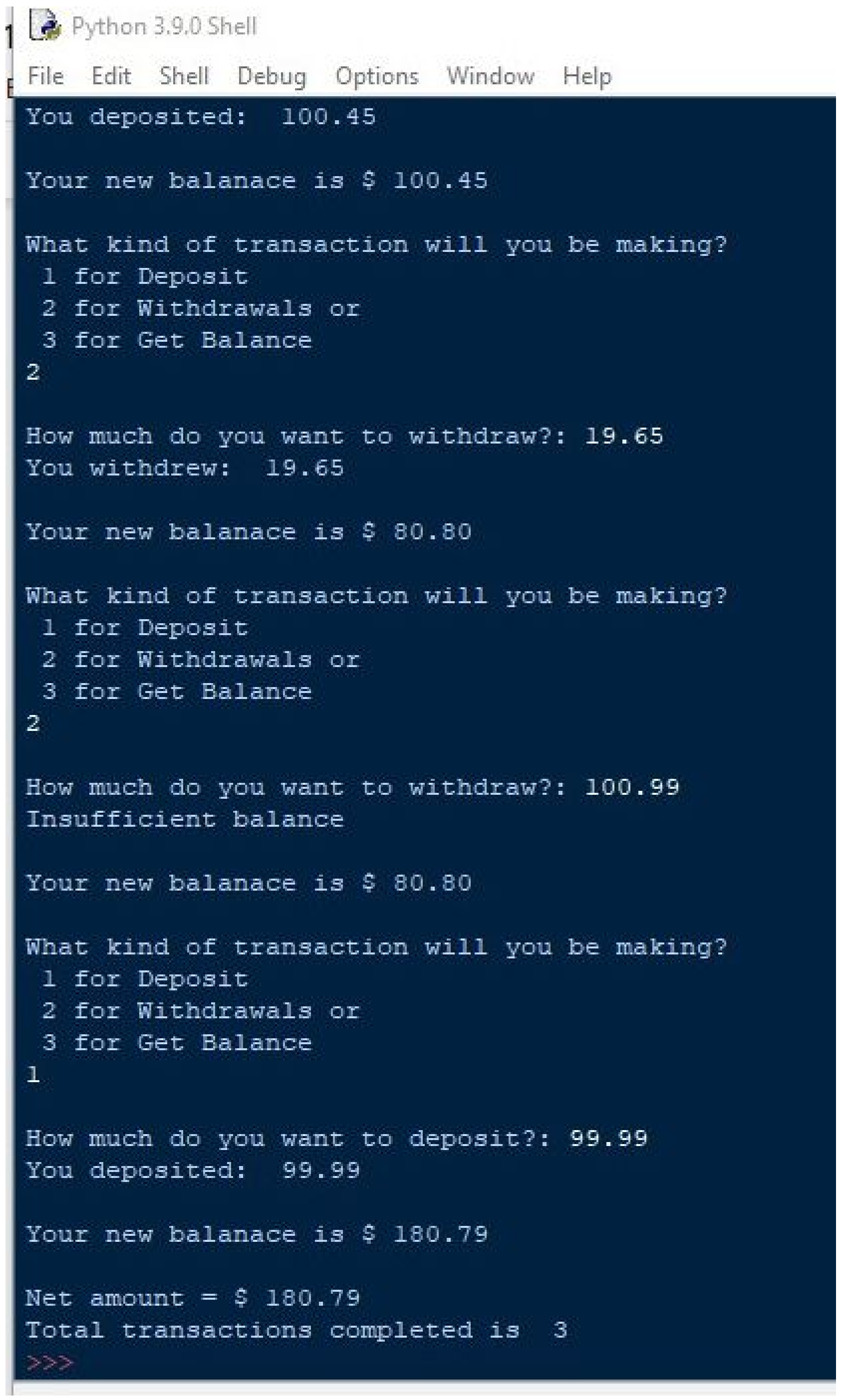
```
Enter number of transactions: 5
Enter transaction type: deposit
Enter transaction amount: -45
Deposit amount $-45.00 is less than 0. Transaction rejected
Enter transaction type: deposit
Enter transaction amount: 100.45
Transaction was successful. Your account balance is $100.45
Enter transaction type: withdraw
Enter transaction amount: 19.65
Transaction was successful. Your account balance is $80.80
Enter transaction type: withdraw
Enter transaction amount: 100.99
Withdraw amount $100.99 is higher than balance of $80.80.
Transaction rejected
Enter transaction type: deposit
Enter transaction amount: 99.99
Transaction was successful. Your account balance is $180.79
After 3 transactions, your balance is: $180.79
```

- If you get the correct result then your program is working as expected.
- **Once you are satisfied with your results a screen shot and paste them below.**

Paste all of your screenshots below this line



```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Users/kerry/Desktop/Register.py ==============
How many transactions will you be making today?: 5

What kind of transaction will you be making?
 1 for Deposit
 2 for Withdrawals or
 3 for Get Balance
1

How much do you want to deposit?: -45
Invalid deposit

Your new balanace is $ 0.00

What kind of transaction will you be making?
 1 for Deposit
 2 for Withdrawals or
 3 for Get Balance
1

How much do you want to deposit?: 100.45
You deposited:  100.45

Your new balanace is $ 100.45

What kind of transaction will you be making?
 1 for Deposit
 2 for Withdrawals or
 3 for Get Balance
2

How much do you want to withdraw?: 19.65
You withdrew:  19.65

Your new balanace is $ 80.80
```



```
You deposited:  100.45

Your new balanace is $ 100.45

What kind of transaction will you be making?
 1 for Deposit
 2 for Withdrawals or
 3 for Get Balance
2

How much do you want to withdraw?: 19.65
You withdrew:  19.65

Your new balanace is $ 80.80

What kind of transaction will you be making?
 1 for Deposit
 2 for Withdrawals or
 3 for Get Balance
2

How much do you want to withdraw?: 100.99
Insufficient balance

Your new balanace is $ 80.80

What kind of transaction will you be making?
 1 for Deposit
 2 for Withdrawals or
 3 for Get Balance
1

How much do you want to deposit?: 99.99
You deposited:  99.99

Your new balanace is $ 180.79

Net amount = $ 180.79
Total transactions completed is  3
>>>
```

- **Upload this lab handout with required screen shots and your code files to Canvas to submit the lab.**