Module Title

Principles of Programming

Weekly Assignment - Practical 6

Year

2025

Student Name: NIRVIK K.C.

UWE ID: 25024649

Assignment Submission Date: January 12, 2026

This assignment consists of the programming questions from practical exercises related to the topics from week 6.

Questions

Part 1 - Python programming

1. Create a new program file

Step 1: Click on practical6 to highlight it and then click on the New File icon. Enter helloworld.py as the name of the file and click on Return. You will see helloworld.py is now opened in the Editor window on the right.

Step 2: Write, compile and run your first Python program

Note that you would need to have Python extension for Visual Studio Code installed. Type the following code in helloworld.py:

```
# Print a message on the screen

print("Hello World! \n")
```

Make sure that you save the file (the black dot on helloworld.py will change to a cross).

Right click your mouse to get a pop-up list, click on Run Code (you can also click on the Run Code arrow on the top right corner) to compile and run your program.

After compiling and running the program, you should get the output shown in the Terminal window below the Editor window. You will see that helloworld.py has been compiled and run. You will also see that Hello World! has been displayed by the program in the Terminal.

Answer:

Following code for input:

```
# read values from input file

# Practical 6 -  Part 1 -  Python programming

# @Nirvik K.C.

# Print a message on the screen

print("Hello World! \n")

# Output: Hello World!
```

Output obtained in execution:

2. (a) Part 2 : Variables, data types, assignments, expressions, operators, comments, and input and output

In this exercise, you are required: first, to create a new Python program file called MonthlyExpenditureA.py in the practical6 folder that you have already created; second, to design, write, compile and run a Python program for calculating the Total Money we spent last month. The program uses three variables as input for Food, Leisure and Clothes expenses, and it uses another variable as output for the total money spent as output.

Step 1 : Click on practical6 in the EXPLORER window, followed by new file to create the MonthlyExpenditureA.py program file in the practical6 folder.

Step 2: Type the following sample code into the program:

```
#Print monthly expenditure

#Practical 6, Part 2 (a)

#author your name

pass
```

This is simply a sample program template and more code will be added to replace pass. Here, pass is a null statement. A pass statement is not ignored by the Python interpreter; however, the statement results into no operation. It is used to create empty code blocks and is useful when you don't write the implementation of a function but you want to implement it in the future.

Step 3: Type, edit and run your program (a). Type the following code into the sample program to replace pass:

Given code:

```
#define float variables and assign values to them
foodExpenses = 300.0 #assign 300.0 to foodExpenses
leisureExpenses = 100.0 #assign 100.0 to leisureExpenses
clothesExpenses = 50.0 #assign 50.0 to clothesExpenses
totalSpent = 0.0 # float variable for total expenses, initialised to 0.0
totalSpent = foodExpenses + leisureExpenses + clothesExpenses
print("The total expenditure this month was: ", totalSpent)
```

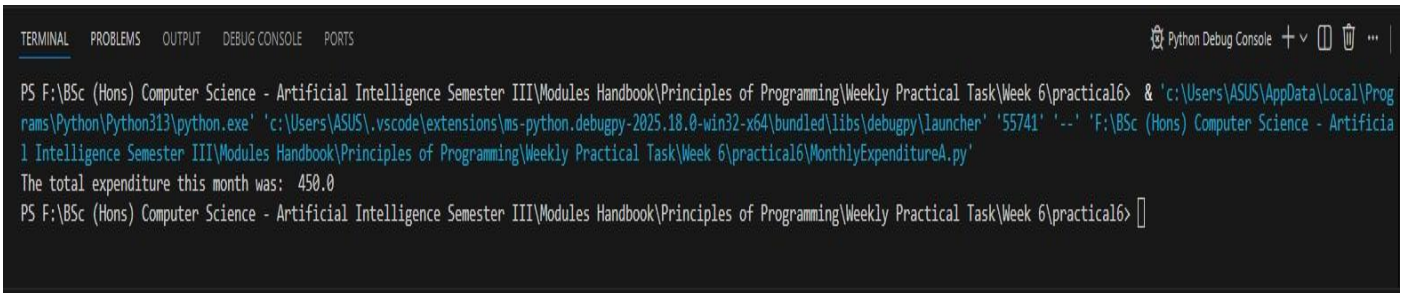
Answer:

Following code for input:

```
# Monthly Expenditure Calculator
# Practical 6, Part 2 (a) Monthly Expenditure
# @Nirvik K.C.
# define float variables and assign values to them
foodExpenses = 300.0    # assign 300.0 to foodExpenses
```

```
leisureExpenses = 100.0 # assign 100.0 to leisureExpenses

clothesExpenses = 50.0  # assign 50.0 to clothesExpenses

totalSpent = 0.0        # assign float variable for total expenses, initialised to 0.0


# Calculate the total expenses

totalSpent = foodExpenses + leisureExpenses + clothesExpenses


# Display the result

print("The total expenditure this month was: ", totalSpent)


# Output: The total expenditure this month was:  450.0
```

Output obtained in execution:



2. (b) Save MonthlyExpenditureA.py as MonthlyExpenditureB.py Extend the program so that it can be used to calculate your total money spent last month on food, leisure, clothes, accommodation and travel. Your new variables for accommodation and travel should be of type integer. Run the extended program.

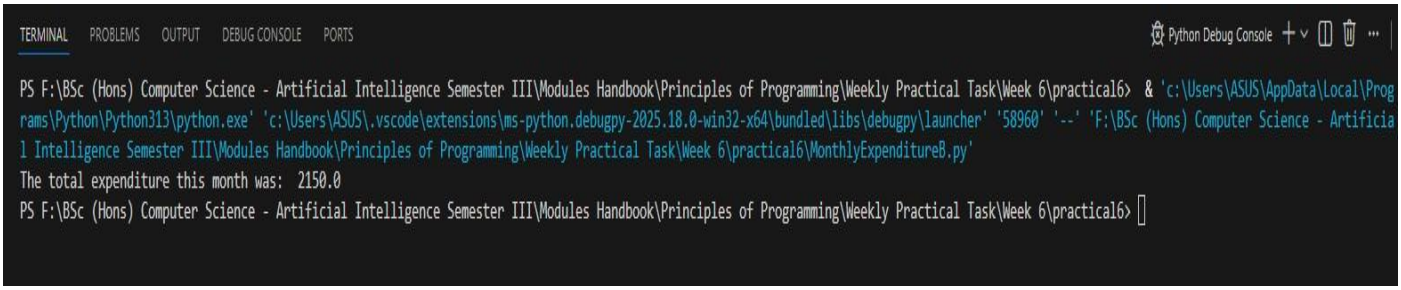
Answer:

Following code for input:

```
# Monthly Expenditure Calculator
# Practical 6, Part 2 (b) Monthly Expenditure
# @Nirvik K.C.
# define float variables and assign values to them
foodExpenses = 300.0     # assign 300.0 to foodExpenses
leisureExpenses = 100.0 # assign 100.0 to leisureExpenses
clothesExpenses = 50.0  # assign 50.0 to clothesExpenses
# new variables for accommodation and travel should be of type integer
accomodationExpenses = 1500     # assign 1500 to accomodationExpenses
travelExpenses = 200    # assign 200 to travelExpenses

totalSpent = 0.0        # assign float variable for total expenses, initialised to 0.0
```

```
# Calculate the total expenses

totalSpent = (foodExpenses + leisureExpenses + clothesExpenses +
accomodationExpenses + travelExpenses)

# Display the result

print("The total expenditure this month was: ", totalSpent)


# Output: The total expenditure this month was:  2150.0
```

Output obtained in execution:



(c) Save MonthlyExpenditureB.py as MonthlyExpenditureC.py. Further extend the program in so that it can be used to take input from the keyboard, calculate your total money spent last month food, leisure, clothes, accommodation and travel, and display your total money on the screen. Run the further extended program.

For this exercise, Python user input from the keyboard can be read using the input() built-in function. After entering the value from the keyboard, we have to press the “Enter” button. Then the input() function reads the value entered by the user.

Here is the sample code for you to get started:

Prompt the user and take input from the user for food expenses foodExpenses = float(input("\n Enter food expenses: "))

Prompt the user and take input from the user for accommodation expenses

accommodationExpenses = int(input("\n Enter accommodation expenses: "))

You can now similarly get input from the user for the other variables.

Answer:

Following code for input:

```
# Monthly Expenditure Calculator

# Practical 6, Part 2 (c) Monthly Expenditure, take input from the keyboard

# @Nirvik K.C.

# Prompt the user and take input from the user for food expenses

foodExpenses = float(input("Enter food expenses: "))
```

```
# Prompt the user and take input from the user for leisure expenses
leisureExpenses = float(input("Enter leisure expenses: "))
# Prompt the user and take input from the user for clothes expenses
clothesExpenses = float(input("Enter clothes expenses: "))
# Prompt the user and take input from the user for accommodation expenses
accomodationExpenses = int(input("Enter accommodation expenses: "))
# Prompt the user and take input from the user for travel expenses
travelExpenses = int(input("Enter travel expenses: "))

totalSpent = 0.0        # assign float variable for total expenses, initialised to 0.0

# Calculate the total expenses
totalSpent = (foodExpenses + leisureExpenses + clothesExpenses +
accomodationExpenses + travelExpenses)

# Display the result
print("The total expenditure this month was: ", totalSpent)
```

Output:

Enter food expenses: 350.50

Enter leisure expenses: 150.00

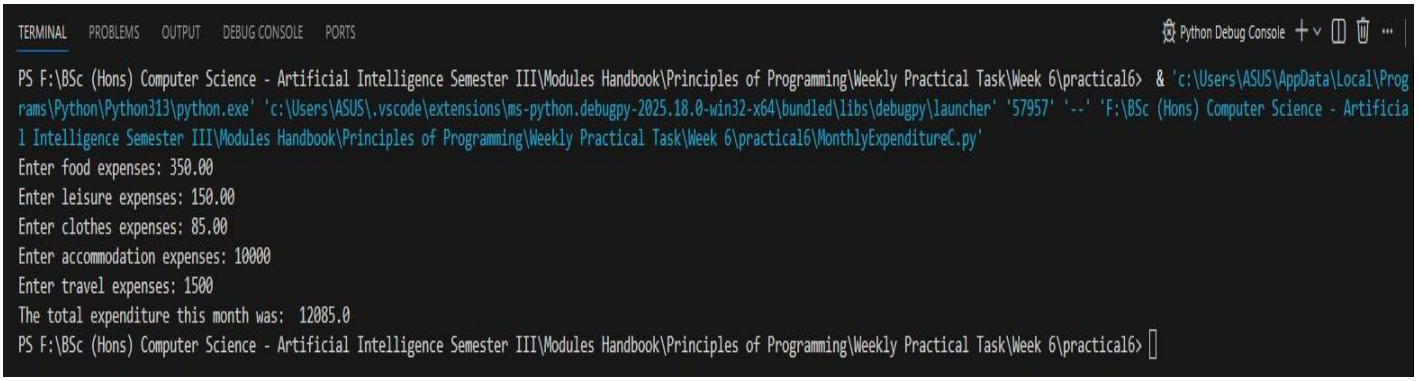
Enter clothes expenses: 85.00

Enter accommodation expenses: 10000

Enter travel expenses: 1500

The total expenditure this month was: 12085.5

Output obtained in execution:



3. Part 3: Selections, Relational Operators and Boolean expressions

Part of the information needed to prepare an electricity bill may be expressed as 4 integers as follows:

previous meter reading

present meter reading

day and month when reading was taken

To be valid the following conditions should be satisfied:

(i) meter readings within range 0..9999

(ii) previous not greater than present

(iii) electricity used not more than 1000 (iv) month in range 1..12, days in month must be correct

In this exercise, you are required: first, to create a program file called ElectricityBillA.py in the practical6 folder that you have already created; second, to design, write and run a Python program which will read input data from the keyboard which consists of 4 integers representing the previous metre reading, the present metre reading, day and month, each separated by a space/return (finish the input with a return).

The program will validate the input data, display the input data and display an appropriate error message for each of the conditions listed above. For example, for the following input data:

900 800 23 5

The program will display:

900 800 23 5

Error: previous reading is more than present reading

Step 1: Add the following program outline

Given code:

```
# Electricity bill
# Practical 6, Part 3
# author your name
# Enter previous meter reading
previousMeterReading = int(input("\n Enter previous meter reading: "))
# Enter current meter reading
# Enter the day of the meter reading
# Enter the month of the meter reading
# Input Validation
```

```
# previous meter reading within range 0..9999

if (previousMeterReading < 0 or previousMeterReading > 9999):

print("Error: previous meter reading out of range \n")

#current meter reading within range 0..9999

# previous not greater than present

# electricity used not more than 1000

# month in range 1..12

# days in month must be correct (Jan, March, etc.)

# days in month must be correct (Apr, June, etc.)

# days in month must be correct (Feb assuming 29 days)
```

Step 2: Run the program outline

Run the above program outline. Though it is a program outline, it contains the program sketch and should be syntactically correct if added correctly.

Step 3: Write the program

Add more code to the program outline to implement your design. Note that some sample code has been provided, which can be used as examples for adding your own code. For reading and displaying the input values, refer to Part 2 in this practical.

Step 4: Test the program

Testing is typically part of program development – you should use a test strategy to test your program thoroughly. For example, the following is a test strategy:

Test Cases		Purpose	Results Expected
0	9999	Test lower and upper range of reading	no error
1	12	Tests lower and upper range of months	no error
0	1000	Tests lower and upper range of units used	no error
1000	890	previous reading is more than present reading	Error message

Test your program with more input data given below. Note that you need to run your program again for each of the following sets of input values (run your program with one set of values at a time, e.g. run it 7 times).

900 800 23 5

9000 9999 12 2

9999 10005 14 6

500 6000 26 7

10000 10100 10 9

6000 6890 30 2

590 829 31 9

Answer:

Following code for input:

```
# Monthly Expenditure Calculator

# Practical 6, Part 3 -  Selections, Relational Operators and Boolean expressions

# @Nirvik K.C.

# Enter previous meter reading

previousMeterReading = int(input("Enter the previous meter reading: "))

# Enter current meter reading

presentMeterReading = int(input("Enter the present meter reading: "))

# Enter the day of the meter reading

day = int(input("Enter the day of the meter reading: "))

# Enter the month of the meter reading

month = int(input("Enter the month of the meter reading (1-12): "))


# Display the input data

print(f"\n{previousMeterReading} {presentMeterReading} {day} {month}")
```

```
# Input validation
# current meter reading within range 0..9999
if (previousMeterReading < 0 or previousMeterReading > 9999):
    print("Error: previous meter reading is out of range\n")

if (presentMeterReading < 0 or presentMeterReading > 9999):
    print("Error: present meter reading is out of range\n")

# Previous reading not greater than present reading
if (previousMeterReading > presentMeterReading):
    print("Error: previous meter reading is more than present meter reading\n")

# Electricity used not more than 1000
usage = presentMeterReading - previousMeterReading
if (usage > 1000):
    print("Error: electricity usage is more than 1000\n")
```

```
#  month in range 1..12
if (month < 1 or month > 12):
    print("Error: month is out of range (must be 1-12)\n")

# Check for months with 31 days
# days in month must be correct (Jan, March, May, July, Augest, October, December)
if month in [1, 3, 5, 7, 8, 10, 12]:
    if day < 1 or day > 31:
        print("Error: day out of range for this month\n")

# Check 30 days in month must be correct (Apr, June, September, November)
elif month in [4, 6, 9, 11]:
    if day < 1 or day > 30:
        print("Error: day out of range for this month\n")
```

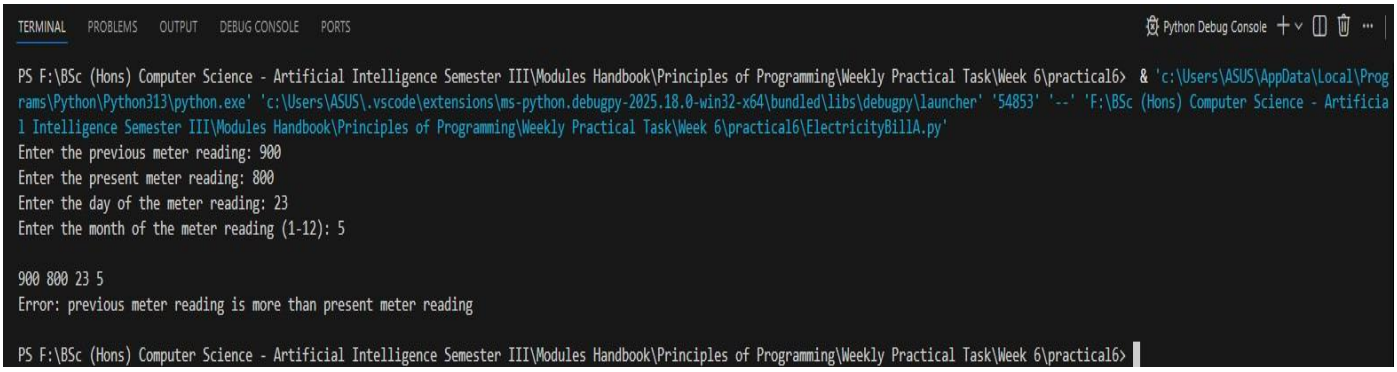
```
# Check separately for February month
elif month == 2:
    if day < 1 or day > 29:
        print("Error: day out of range for February\n")



# Sample Output:
# Enter the previous meter reading: 900
# Enter the present meter reading: 800
# Enter the day of the meter reading: 23
# Enter the month of the meter reading (1-12): 5


# 900 800 23 5
# Error: previous meter reading is more than present meter reading
```

Test Cases:

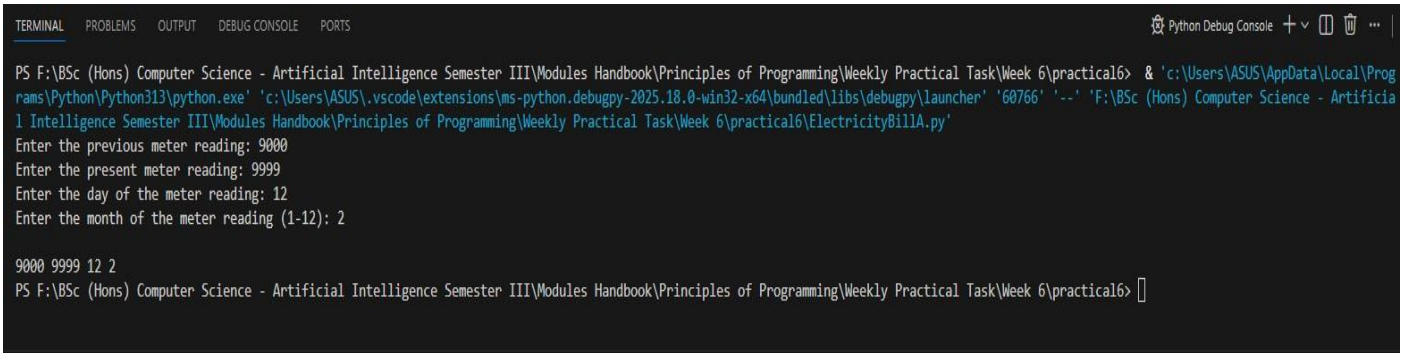
Test Cases	Purpose	Results Expected
900 800 23 5	Test previous reading greater than present reading	900 800 23 5 Error: previous meter reading is more than present meter reading
9000 9999 12 2	Test upper range of valid meter readings + February	9000 9999 12 2 (No error - valid input)
9999 10005 14 6	Test upper limit of present reading + day validation	9999 10005 14 6 Error: present meter reading out of range
500 6000 26 7	Test upper limit of units used (>1000)	500 6000 26 7 Error: electricity usage is more than 1000
10000 10100 10 9	Test upper limit violation of previous meter reading (>9999)	10000 10100 10 9 Error: previous meter reading out of range Error: present meter reading out of range
6000 6890 30 2	Test upper limit of days in February	6000 6890 30 2 Error: day out of range for February
590 829 31 9	Test upper limit of days in 30-day month	590 829 31 9 Error: day out of range for this month

Output obtained in execution:

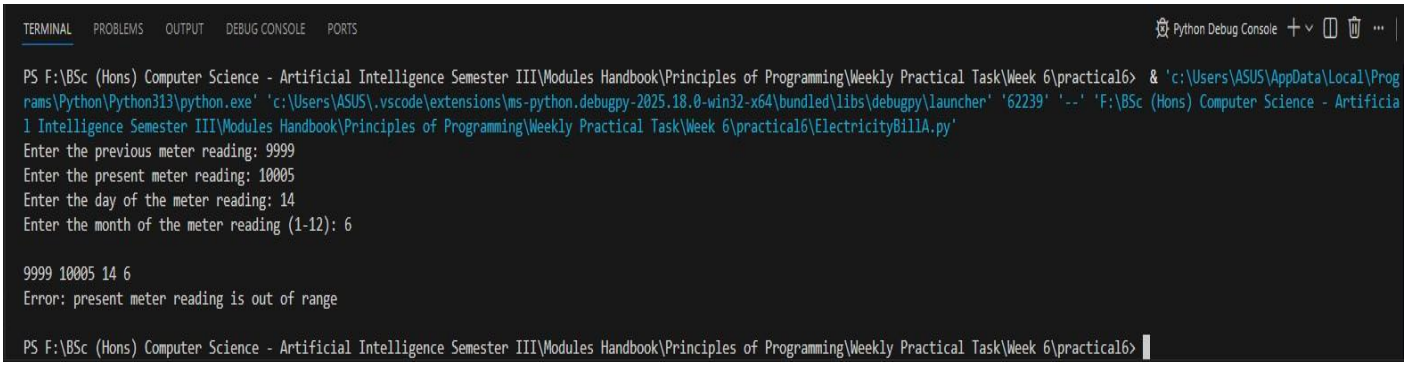
Test Case 1:



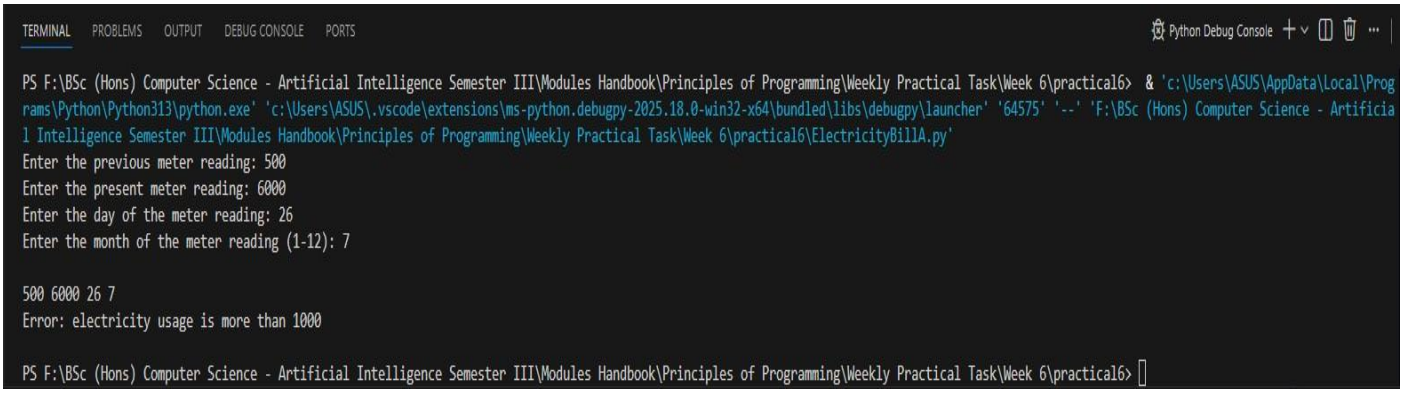
Test Case 2:



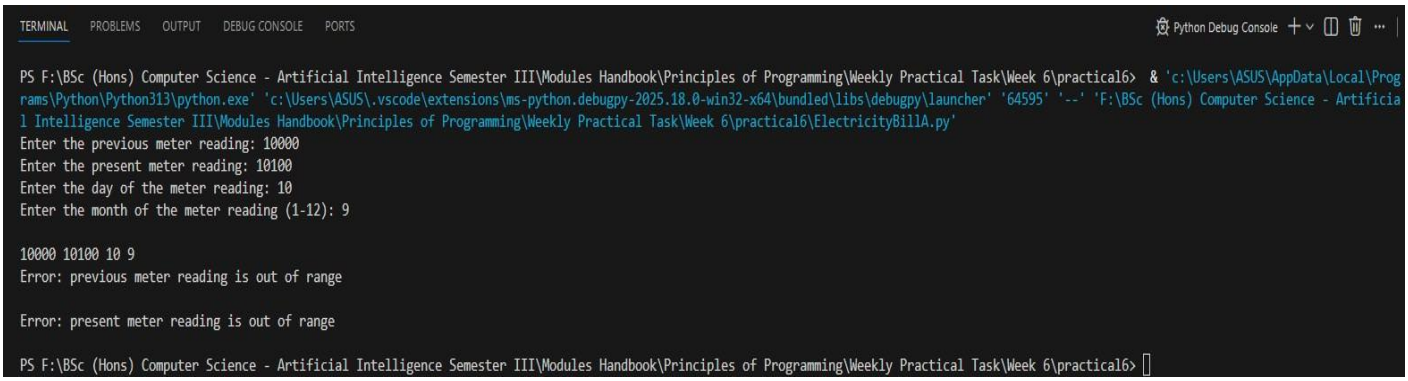
Test Case 3:



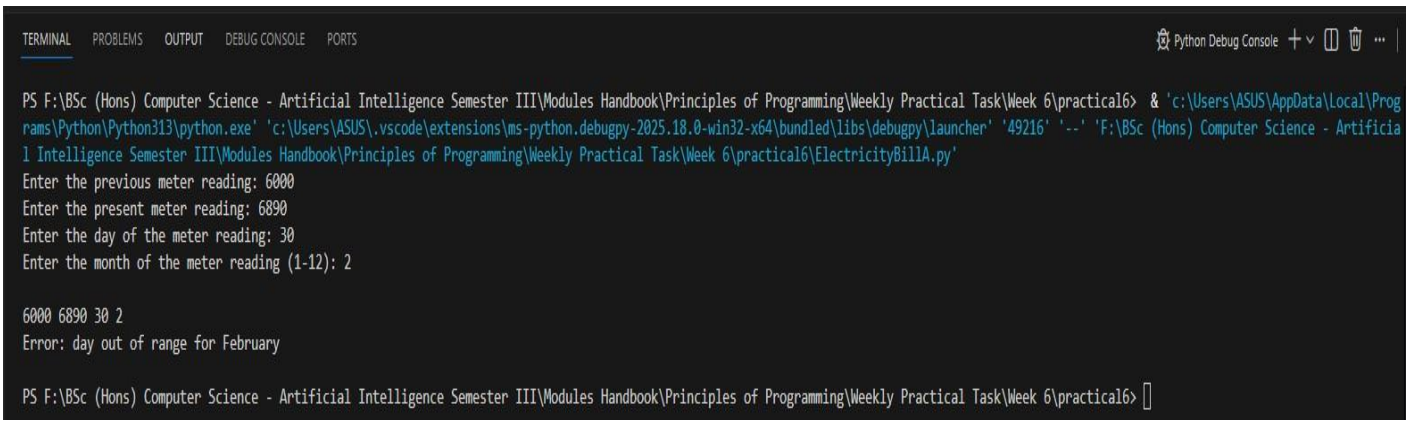
Test Case 4:



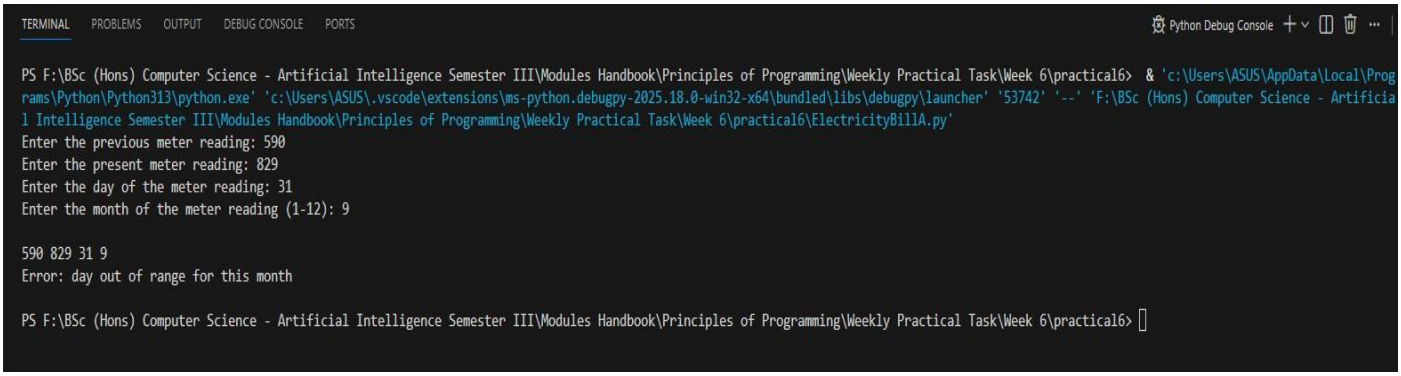
Test Case 5:



Test Case 6:



Test Case 7:



4. Part 4: Loops, lists and functions

Like C, Python supports for and while loops. Loops are used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

Exercise 1. In this exercise, you are required: first, to create a program file called loopTest.py in the practical6 folder that you have already created; second, type the following code, save and run it. Given code:

#Printing months from a list using a for loop

#Practical 6, Part 4

#author your name months = [“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, “Dec”]

for x in months:

  print(x)

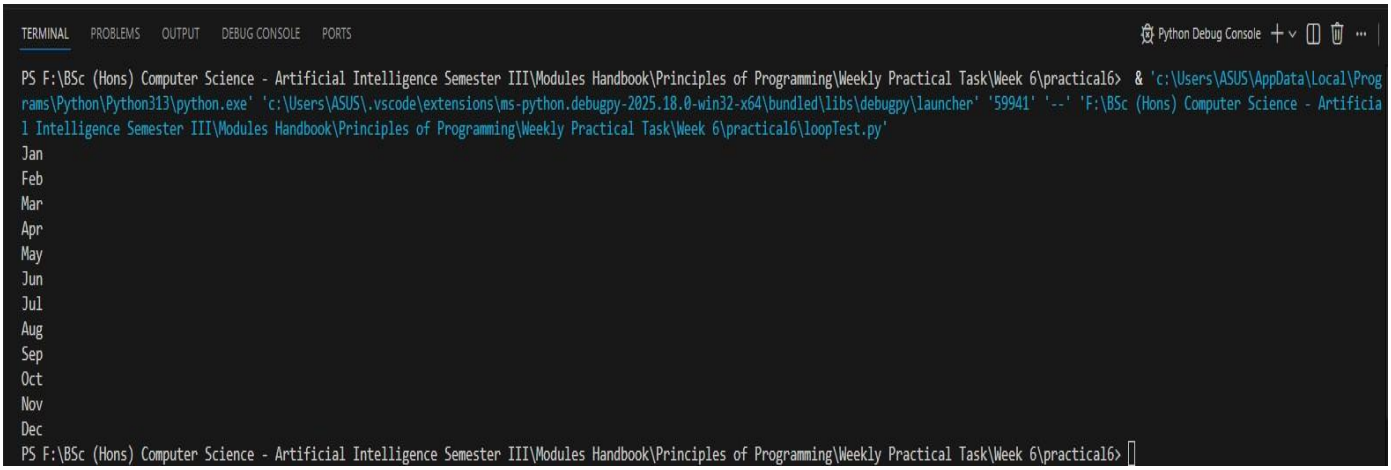
Answer:

Following code for input:

```
# Printing months from a list using a for loop

# Practical 6, Part 4 - Exercise 1

# @Nirvik K.C.

months = [“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, “Dec”]

for x in months:

    print(x)      # print each month in the list


# Output:

# Jan

# Feb

# Mar

# Apr

# May

# Jun

# Jul
```

```
# Aug

# Sep

# Oct

# Nov

# Dec
```

Output obtained in execution:



Exercise 2. With the continue statement we can stop the current iteration of the loop, and continue with the next. For example, if the month is Apr we can skip printing it. Modify your program in Exercise 1 as below, save and run it.

25024649

Given code:

```
#Printing months from a list using a for loop

#Practical 6, Part 4

#author your name

months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

for x in months:

    if x == "Apr":

      continue

    print(x)
```

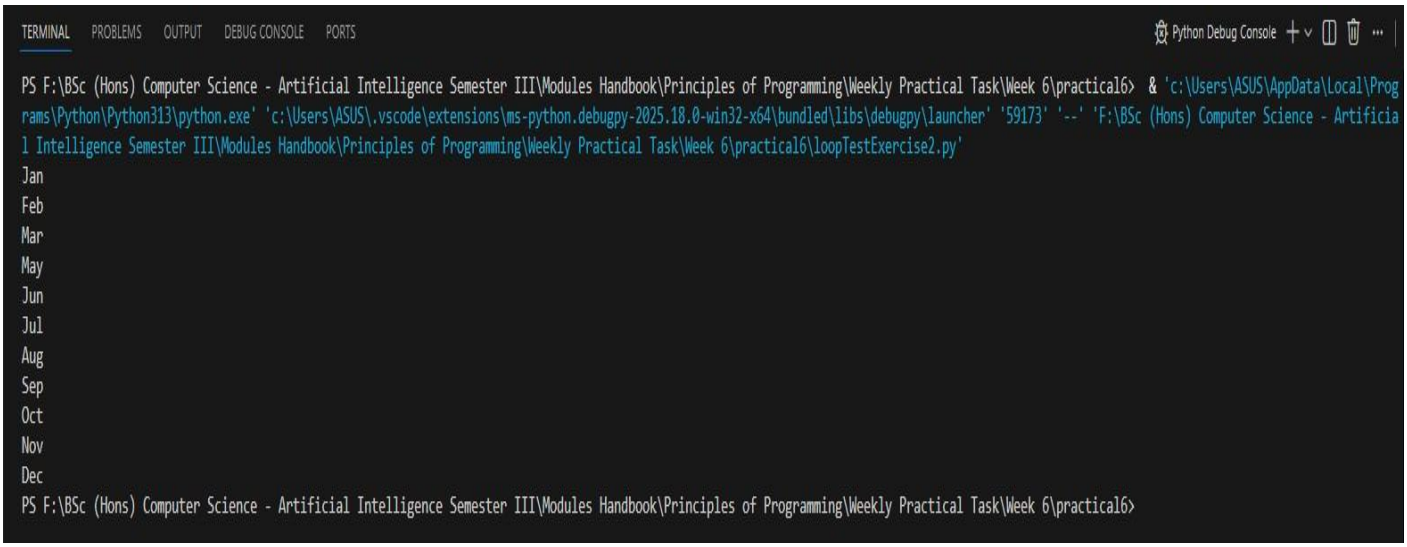
Answer:

Following code for input:

```
# Printing months from a list using a for loop

# Practical 6, Part 4 - Exercise 2

# @Nirvik K.C.
```

```
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct",
"Nov", "Dec"]

for x in months:

    if x == "Apr":

        continue

    print(x)  # continue statement skips the print(x) line for the month of April

# Output:

# Jan

# Feb

# Mar

# May

# Jun

# Jul

# Aug

# Sep

# Oct

# Nov

# Dec
```

Output obtained in execution:



Exercise 3. We can use the range() function to loop a given number of times through a set of code. The range() function returns a sequence of numbers which, by default, starts from 0 and increases by 1 (by default), and ends at the number defined.

Create a program file called loopTest2.py in the practical6 folder that you have already created, type the following code, save and run it.

#Using range() function in a for loop

#Practical 6, Part 4

#author your name

```
for x in range(10):
  print(x)
```

Answer:

Following code for input:

```
# Using range() function in a for loop
# Practical 6, Part 4 - Exercise 3
# @Nirvik K.C.

for x in range(10):
    print(x)  # The for loop runs 10 times, printing each number on a new line

# Output:
# 0
# 1
# 2
# 3
# 4
```

5

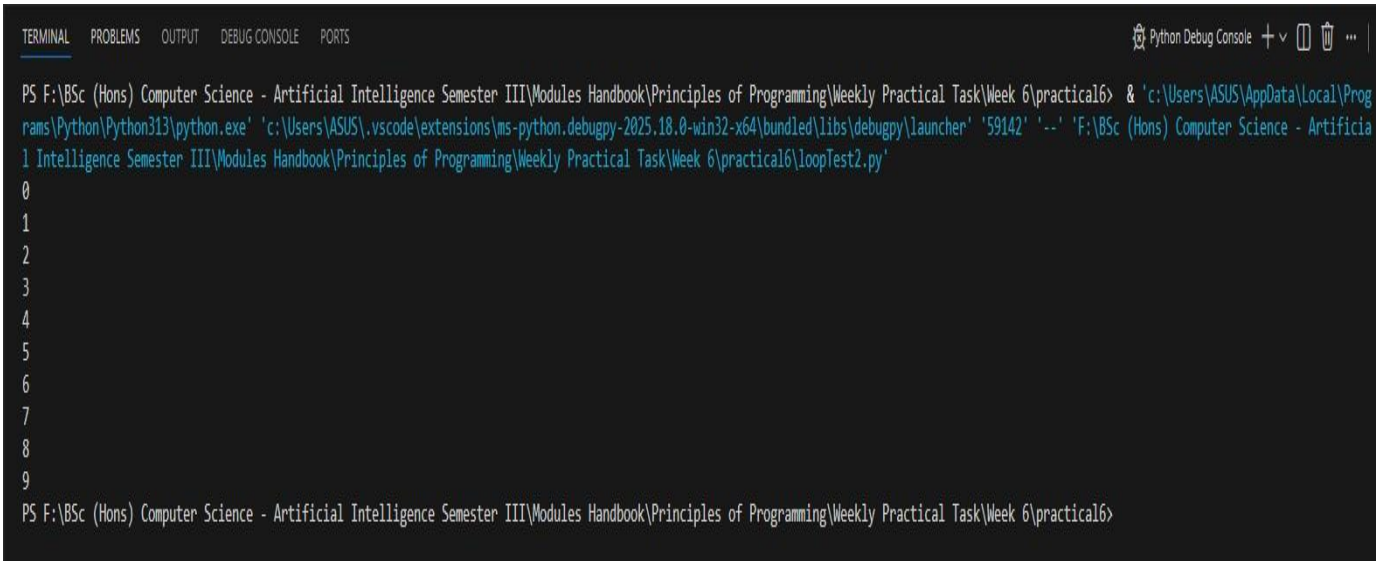
6

7

8

9

Output obtained in execution:



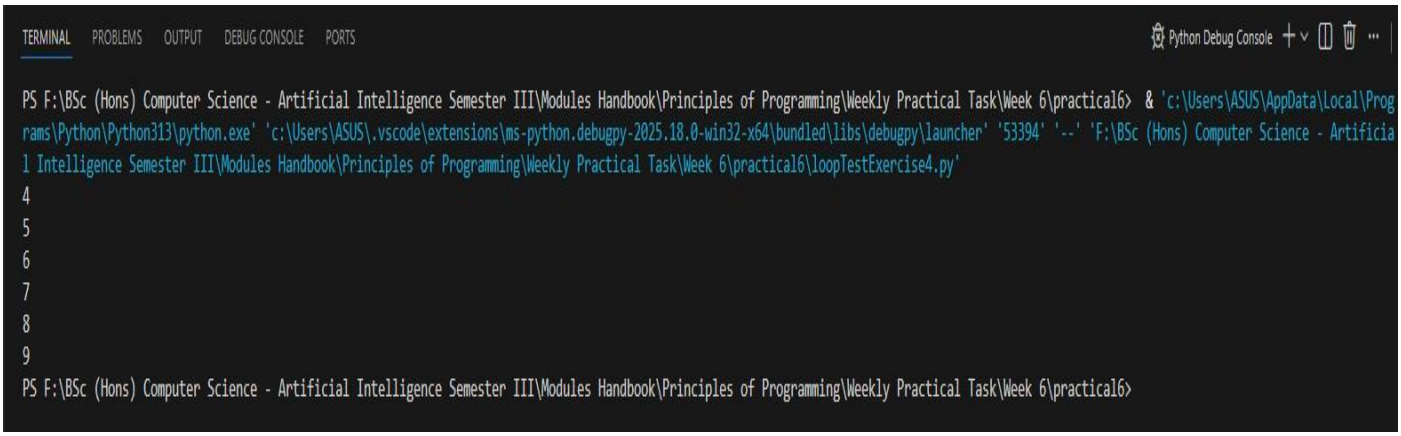
Exercise 4. The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(4, 10), which means values from 4 to 10 (but not including 6). Replace range(10) by range(4,10)in your Exercise 3, save and run your code.

Answer:

Following code for input:

```
# Using range() function in a for loop

# Practical 6, Part 4 - Exercise 4

# @Nirvik K.C.

for x in range(4, 10):

    print(x)   # The for loop runs from 4 to 9, printing each number on a new line


# Output:

# 4

# 5

# 6

# 7

# 8

# 9
```

Output obtained in execution:



Exercise 5. Like C, a nested loop in Python is a loop inside a loop. The inner loop will be executed one time for each iteration of the outer loop.

Create a program file called loopTest3.py in the practical6 folder that you have already created, type the following code, save and run it.

```
#Nested loops

#Practical 6, Part 4

#author your name

for x in range(10):

 for y in range (5):

  print(x,y)
```

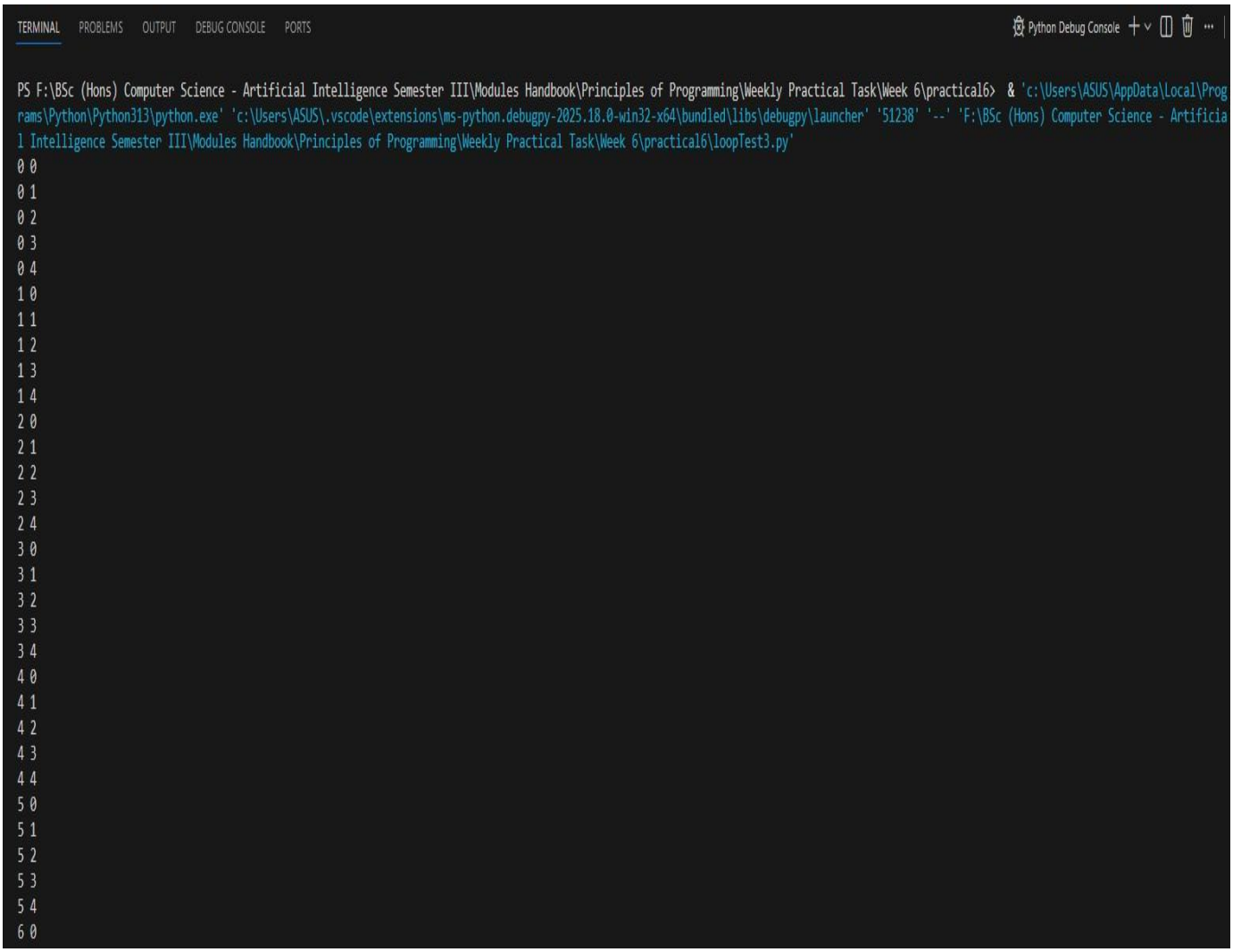
Answer:

Following code for input:

```
# Nested loops

# Practical 6, Part 4 - Exercise 5

# @Nirvik K.C.

for x in range(10):

    for y in range(5):

        print(x,y)  # The outer loop runs 10 times, and for each iteration of the outer loop, the inner loop runs 5 times


# Output: The program will print a total of 50 lines (10 * 5 = 50)
```

Output obtained in execution:

First picture:



Continue:

Second picture:

Exercise 6. In Python, a list is a collection which is ordered and changeable, and is written with square brackets. In Exercise 2 above months is a list:

months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

You can access the list items by referring to the index number.

For example, print(months[0]) will print Jan, print(months[1]) will print Feb, print(months[2]) will print Mar and so on.

Create a program file called listTest.py in the practical6 folder that you have already created, type the following code, save and run it.

Given code:

```
# Entering and Printing months from a list using while and for loops

# Practical 6, Part 4

# author your name

months = []


index = 0

while index < 12:

    m = input("Enter a month: ")

    months.append(m)

    index += 1


print("Now printing the months you entered\n")

for x in range(12):

   print(months[x])
```

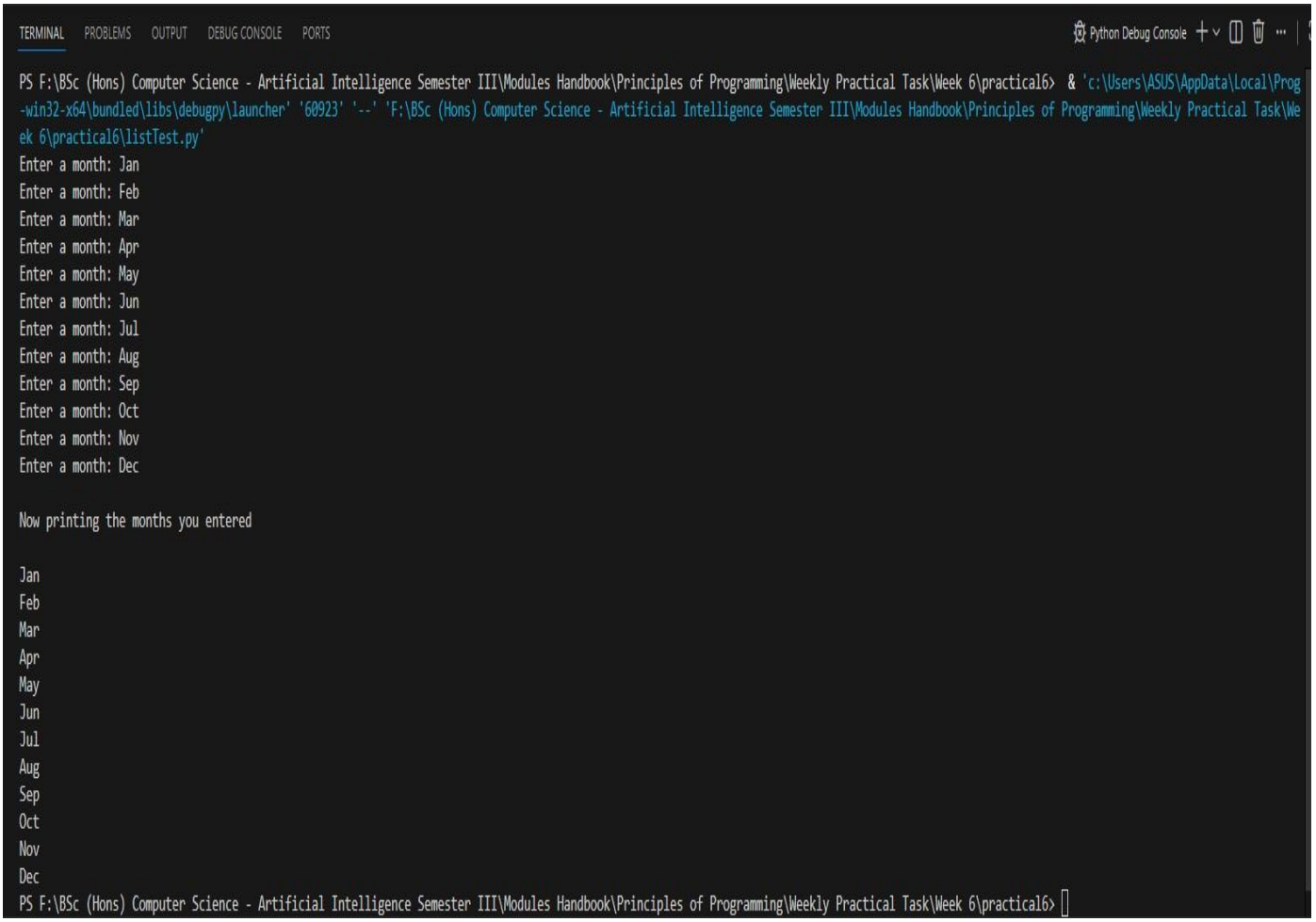
Answer:

Following code for input:

```
# Access the list items by referring to the index number
# Practical 6, Part 4 - Exercise 6
# @Nirvik K.C.
# Using the already defined months list
months = []
index = 0

# Loop through the list using index
while index < 12:
    m = input("Enter a month: ")
    months.append(m)  # Append the user input to the months list
    index += 1        # Increment the index
print("\nNow printing the months you entered\n")
for x in months:
    print(x)  # print each month in the list
```

Output: Prints the months entered by the user

Output obtained in execution:



Exercise 7. In a criminal investigation, DNA profiling was used to match the DNA of a suspect who has been arrested for suspicion of committing a burglary against the DNA profiling of several criminals who have been convicted of committing similar crimes and therefore identified as a repeated offender. To identify a repeated offender correctly 10 particular gene chromosomes of the suspect's DNA profile were chosen which must exactly match for the corresponding ones of a convicted criminal.

For example, the suspect's DNA profile consists of 10 real numbers as follows:

2.3 3.3 4.5 6.7 7.8 2.1 3.2 4.3 5.2 6.5

and the DNA profile of a criminal is as follows:

2.3 3.3 4.5 6.7 7.8 2.1 3.2 4.3 5.2 6.5

In this case, two profiles match so the suspect is a repeated offender.

Create a new Python program in the practical6 folder, called MatchingProfilesA.py. The program contains two lists, each of which has 10 elements representing the DNA profile of a suspect and the one of a convicted criminal. The program read the two profiles from the keyboard (one element at a time), match them to decide whether they match, and display the matching result on the screen.

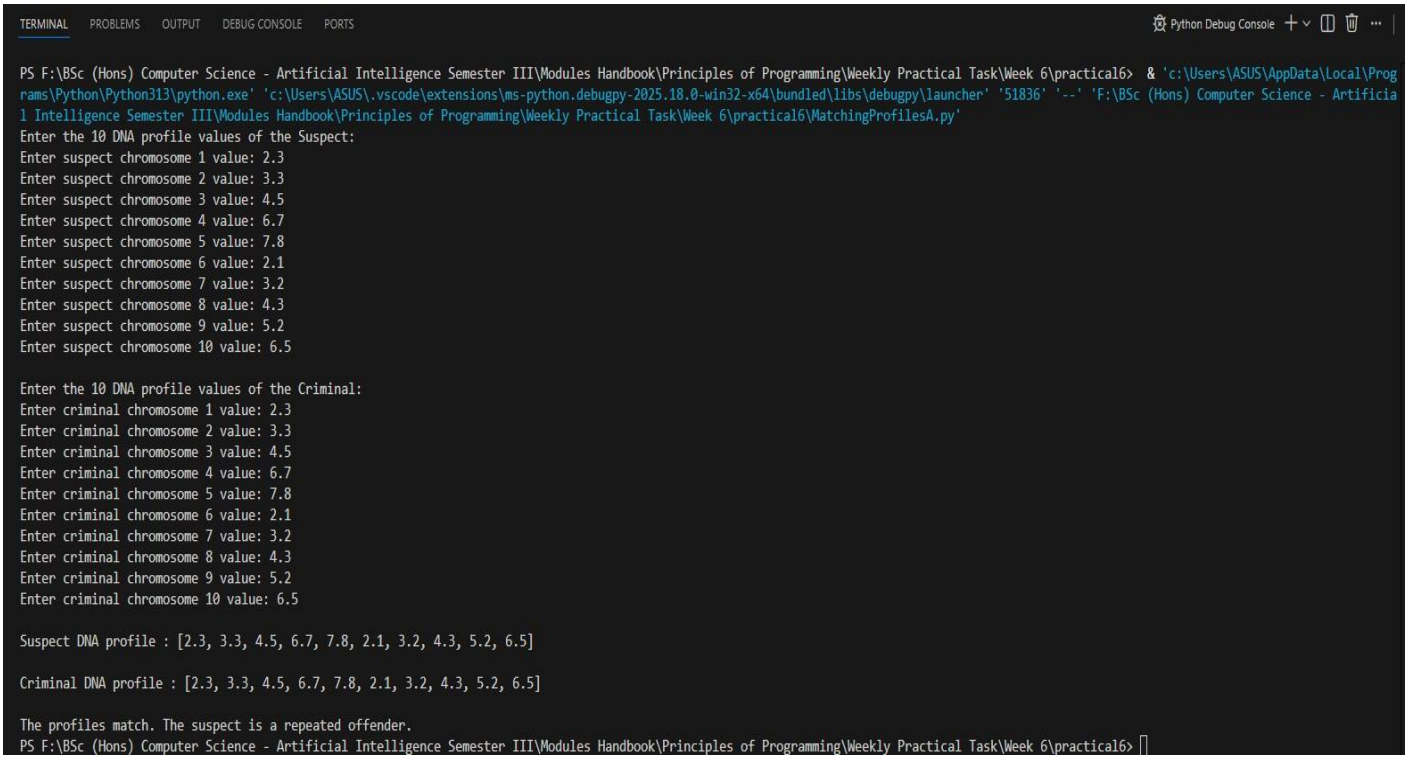
Answer:

Following code for input:

```
# DNA Profile Matching - Identify a repeated offender
# Practical 6, Part 4 - Exercise 7
# @Nirvik K.C.
# Two empty lists to store DNA profiles of Suspects and Criminal
suspect_profiles = []
criminal_profiles = []
# Enter DNA profiles for Suspect
print("Enter the 10 DNA profile values of the Suspect: ")
for i in range(10):
    suspect_gene = (float(input(f"Enter suspect chromosome {i+1} value: ")))
    suspect_profiles.append(suspect_gene)
# Enter DNA profiles for Criminal
print("\nEnter the 10 DNA profile values of the Criminal: ")
for i in range(10):
    criminal_gene = float(input(f"Enter criminal chromosome {i+1} value: "))
    criminal_profiles.append(criminal_gene)
```

```
# Compare DNA profiles
print("\nSuspect DNA profile :",suspect_profiles)
print("\nCriminal DNA profile :", criminal_profiles)
if suspect_profiles == criminal_profiles:
    print("\nThe profiles match. The suspect is a repeated offender.")
else:
    print("\nThe profiles do not match.")
```

Output obtained in execution:



Exercise 8. Consider the criminal investigation problem in Exercise 7 above, modify your code to use a function matchingProfiles() that takes two lists as argument and returns True if two profiles match so the suspect is a repeated offender. You may create a new Python program in the practical6 folder, called MatchingProfilesB.py.

Answer:

Following code for input:

```
# DNA Profile Matching - Identify a repeated offender with Function
# Practical 6, Part 4 - Exercise 8
# @Nirvik K.C.
def matchingProfiles(suspect_dna, criminal_dna):
    # Th function compares the two list of DNA profiles and returns True if they match, otherwise False
    if suspect_dna == criminal_dna:
        return True
    else:
        return False
```

```
# Two empty lists to store DNA profiles of Suspects and Criminal
suspect_profiles = []
criminal_profiles = []

# Enter DNA profiles for Suspect
print("Enter the 10 DNA profile values of the Suspect: ")
for i in range(10):
    suspect_gene = float(input(f"Enter suspect chromosome {i+1} value: "))
    suspect_profiles.append(suspect_gene)

# Enter DNA profiles for Criminal
print("\nEnter the 10 DNA profile values of the Criminal: ")
for i in range(10):
    criminal_gene = float(input(f"Enter criminal chromosome {i+1} value: "))
    criminal_profiles.append(criminal_gene)

# The function to check the match
is_repeated_offender = matchingProfiles(suspect_profiles, criminal_profiles)
```

```
# Display the result
print("\nSuspect DNA profile :",suspect_profiles)
print("\nCriminal DNA profile :", criminal_profiles)


if is_repeated_offender:
    print("\nThe profiles match. The suspect is a repeated offender.")
else:
    print("\nThe profiles do not match.")
```

Output obtained in execution: