# Module 5

**Part 1:**

Write a program that uses nested loops to collect data and calculate the average rainfall over a period of years. The program should first ask for the number of years. The outer loop will iterate once for each year. The inner loop will iterate twelve times, once for each month. Each iteration of the inner loop will ask the user for the inches of rainfall for that month. After all iterations, the program should display the number of months, the total inches of rainfall, and the average rainfall per month for the entire period.

**Solution**:

1. Create a dictionary object for displaying user friendly data for month information.

dict\_month = { '01':'January',

'02':'February',

'03':'March',

'04':'April',

'05':'May',

'06':'June',

'07':'July',

'08':'August',

'09':'September',

'10':'October',

'11':'November',

'12':'December' }

1. Define MAX\_YEARS = 100 as a maximum number of years to avoid an infinite loop.
2. Create a list object to capture and store rainfall data inserted by the user.
3. Create a counter object to calculate the total number of months.
4. Define all the application flow within try/except block for better error handling and avoid application crash instances (geeksforgeeks, 2023).
5. Get a user input on the number of years for processing and validate user input to be integer within 0 to 100. Show an appropriate validation message for incorrect range. Usage of chaining comparison operators : ***0 < number\_of\_years < MAX\_YEARS*** (geeksforgeeks, 2024)
6. Logic - Here, I am using a nested loop. Outer loop runs for a number of years and the inner loop will run for 12 times(for a number of months) (Gaddis, 2017).

for year in range(1, number\_of\_years+1):

for month in dict\_month.values():

print('Please Enter Rainfall for Year:', year, 'and Month:', month)

inches\_rain = float(input())

lst\_rainfall.append(inches\_rain)

total\_number\_of\_months += 1

1. For each inner loop condition, the program accepts a user input (float) for getting the value of rainfall for each month for the index of the outer loop representing the year.
2. Program append each user inserted value into lst\_rainfall and increment month counter.
3. Program displays the total number of months.
4. Program calculates the total number of rainfall using ***sum()***and uses ***round()*** to round up the output to 2 decimal points.

print('Total Number of Months:', total\_number\_of\_months)

print('Total rainfall (in', number\_of\_years, 'year(s)):', round((sum(lst\_rainfall)),2), 'inch(es)')

print('Average rainfall:', round((statistics.mean(lst\_rainfall)),2), 'inch(es) per month \n')

1. Program uses a **statistics** module to calculate the **mean** value of rainfall. The **statistics** module/library has been imported at the beginning of the program (Python, n.d.).

Alternatively - Program can do calculation using:

*if len(lst\_rainfall) > 0 :*

*Avg\_rain = sum(lst\_rainfall)/len(lst\_rainfall)*

1. Display user friendly output.

**Program Code:**

import statistics

dict\_month = { '01':'January',

'02':'February',

'03':'March',

'04':'April',

'05':'May',

'06':'June',

'07':'July',

'08':'August',

'09':'September',

'10':'October',

'11':'November',

'12':'December' }

MAX\_YEARS = 100

lst\_rainfall =[]

total\_number\_of\_months = 0

try:

number\_of\_years = int(input('Please Enter number of Years to calculate Rainfall Stats.\n'))

if(0 < number\_of\_years < MAX\_YEARS):

for year in range(1, number\_of\_years+1):

for month in dict\_month.values():

print('Please Enter Rainfall for Year:', year, 'and Month:', month)

inches\_rain = float(input())

lst\_rainfall.append(inches\_rain)

total\_number\_of\_months += 1

else:

print("Invalid Entry. (Please try again and enter whole number between 0 and 100.)")

print('\t \_\_ \_')

print('\t \_( )\_( )\_')

print('\t (\_ \_ \_)')

print('\t / /(\_) (\_\_)')

print('\t / / / / / /')

print('\t/ / / / / /')

print('===== Rain Statistics =====\n')

print('Total Number of Months:', total\_number\_of\_months)

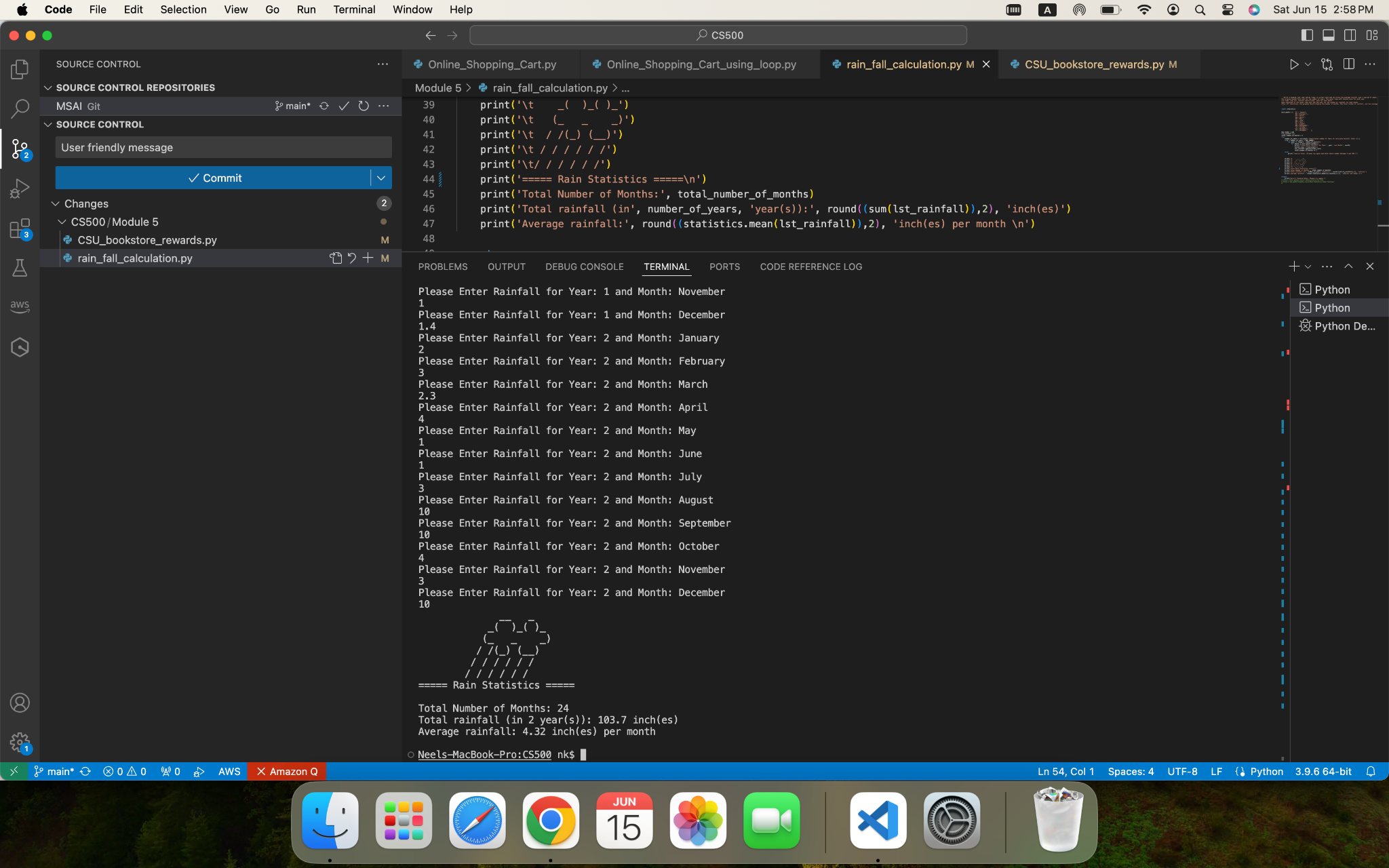
print('Total rainfall (in', number\_of\_years, 'year(s)):', round((sum(lst\_rainfall)),2), 'inch(es)')

print('Average rainfall:', round((statistics.mean(lst\_rainfall)),2), 'inch(es) per month \n')

except:

print('Error!! Invalid Entry. Please try again.')

**Output:**



**GITHub:**

<https://github.com/nshahcsu/MSAI/blob/main/CS500/Module%205/rain_fall_calculation.py>

**Part 2:**

The CSU Global Bookstore has a book club that awards points to its students based on the number of books purchased each month. The points are awarded as follows:

* If a customer purchases 0 books, they earn 0 points.
* If a customer purchases 2 books, they earn 5 points.
* If a customer purchases 4 books, they earn 15 points.
* If a customer purchases 6 books, they earn 30 points.
* If a customer purchases 8 or more books, they earn 60 points.

Write a program that asks the user to enter the number of books that they have purchased this month and then display the number of points awarded.

**Solution**:

1. Define method for calculating rewards points. It accepts one parameter - number of books purchased this month (geeksforgeeks, 2024).
2. Define if elif blocks for translating reward structure (Gaddis, 2017).

Here, the condition says, the customer will get 0 points for 0 books, 5 points for 2 books, 15 points for 4 books etc. In this rewards structure, I have considered following logic. As an example, if a customer buys 3 books. There is no condition specified in the problem statement for 3 books. In this case, it will automatically fall to the next best category i.e. 5 reward points for purchasing 2 books. This can be achieved by multiple ways:

**Option 1:**

Bottom to top approach (usage of condition check for range of values)

def calculate\_points(books\_this\_month):

if(0 < books\_this\_month < 2):

return 0

elif(2 <= books\_this\_month < 4):

return 5

elif(4 <= books\_this\_month < 6):

return 15

elif(6 <= books\_this\_month < 8):

return 30

elif(books\_this\_month >= 8):

return 60

else:

print('Error: Invalid Entry.')

**Option 2:**

Top to bottom approach (Compare the highest tier first, if customer matches that, assign rewards and return value. If not, compare the next best tier and assign reward points accordingly)

def calculate\_points(books\_this\_month):

if(books\_this\_month >= 8):

return 60

elif(books\_this\_month >= 6):

return 30

elif(books\_this\_month >= 4):

return 15

elif(books\_this\_month >=2):

return 5

elif(books\_this\_month >=0):

return 0

else:

print('Error: Invalid Entry.')

1. Program uses try/except blocks for unexpected error handling scenarios(geeksforgeeks, 2023).
2. Program accepts integer value as an input value (number of books purchased this month)
3. Once a user enters a value, the program invokes the method to calculate rewards and return the value.
4. Program displays user-friendly output with rewards information.

**Program Code:**

def calculate\_points(books\_this\_month):

if(books\_this\_month >= 8):

return 60

elif(books\_this\_month >= 6):

return 30

elif(books\_this\_month >= 4):

return 15

elif(books\_this\_month >=2):

return 5

elif(books\_this\_month >=0):

return 0

else:

print('Error: Invalid Entry.')

try:

i\_books\_this\_month = int(input('Please Enter number of books purchased this month:'))

points\_earned = calculate\_points(books\_this\_month=i\_books\_this\_month)

print('\t ,')

print('\t\_\_/ \\_\_')

print('\t\ /')

print('\t/\_ \_\\')

print('\t \\ /')

print('===== Rewards Summary ======\n')

print('Number of Books Purchased :', i\_books\_this\_month)

print('Points Earned for this month:', points\_earned )

if(i\_books\_this\_month):

print('\nEnjoy Reading the books! Thank you!\n')

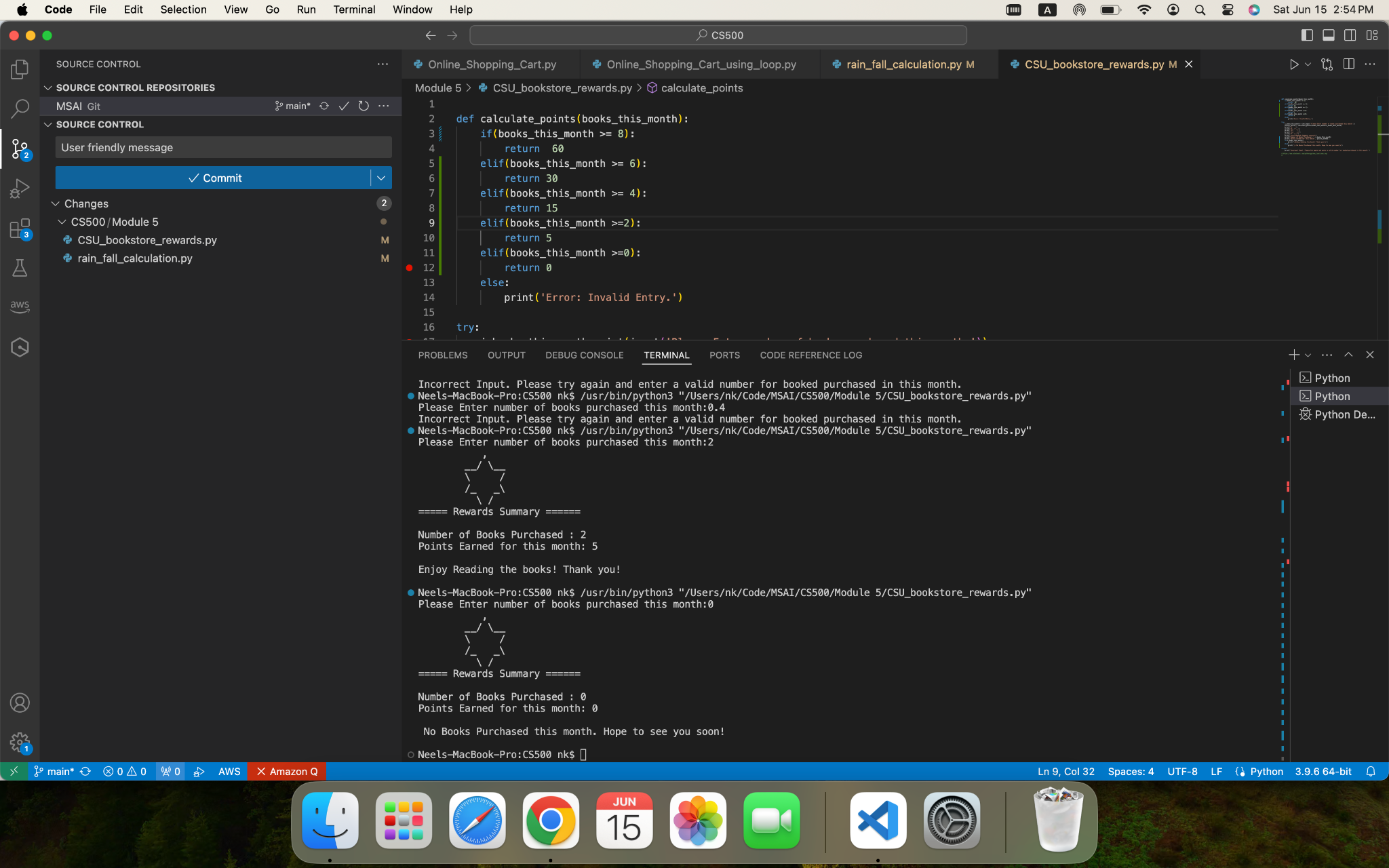
else:

print('\n No Books Purchased this month. Hope to see you soon!\n')

except:

print('Incorrect Input. Please try again and enter a valid number for booked purchased in this month.')

**Output:**



**GITHub:**

<https://github.com/nshahcsu/MSAI/blob/main/CS500/Module%205/CSU_bookstore_rewards.py>

**References:**

Gaddis, T. (2017). Starting Out with Python (4th Edition).

GeeksforGeeks. (2024, May 3). Comparison Operators in Python. GeeksforGeeks. https://www.geeksforgeeks.org/relational-operators-in-python/

GeeksforGeeks. (2024, April 5). Python Functions. GeeksforGeeks. <https://www.geeksforgeeks.org/python-functions/>

GeeksforGeeks. (2023, April 13). Python try except. GeeksforGeeks. https://www.geeksforgeeks.org/python-try-except/

Python, R. (n.d.). Math and statistics functions. https://realpython.com/lessons/math-and-statistics-functions/