



**PANGASINAN STATE UNIVERSITY**

URDANETA CITY CAMPUS

SECOND SEMESTER, AY 2021-2022

LABORATORY

REPORT

CpE222 SOFTWARE DESIGN

**SUBMITTED BY**

**Lendl Johnfil M. Claro**

**SUBMITTED TO**

**Engr. Jay-Ar M. Pentecostes**

**Laboratory Activity 04**

(PYTHON) Content:

1. PreLab

1.1 Recursion:

• Chapters 4: Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al

Write your own Observation and Conclusion from what you have read.

Recursion is a technique by which a function makes one or more calls to itself during execution, or by which a data structure relies upon smaller instances of the very same type of structure in its representation. We have 3 types of recursions: Linear, Binary and Multiple. If a recursive function is designed so that each invocation of the body makes at most one new recursive call, this is known as linear recursion. When a function makes two recursive calls, we say that it uses binary recursion. We define multiple recursions as a process in which a function may make more than two recursive calls.

1.2 Recursion: Part I p.4 to p.69: Ref Lab4 The Recursive Book of Recursion Ace the Coding Interview with Python and JavaScript (Early Access), Al Sweigart

Write your own Observation and Conclusion from what you have read.

Recursion has an intimidating reputation. It’s considered hard to understand, but at its core, it depends on only two things: function calls and stack data structures. Functions can be described as mini-programs inside your program. They’re a feature of nearly every programming language. If you need to run identical instructions at three different places in a program, instead of copying and pasting the source code three times, you can write the code in a function once and call the function three times. A stack is one of the simplest data structures in computer science. It stores multiple values like a list does—but unlike lists, it limits you to adding to or removing values from the “top” of the stack only. For lists, the “top” is the last item, at the right end of the list. Adding values is called pushing values onto the stack, while removing values is called popping values off the stack. A recursive function is a function that calls itself.

This chapter explored several algorithms that make use of tree data structures and backtracking, which are features of a problem that make it suitable for solving with recursive algorithms. We covered tree data structures, which are composed of nodes that contain data and edges that relate nodes together in parent-child relationships. In particular, we examined a specific kind of tree called a directed acyclic graph (DAG) that is often used in recursive algorithms. A recursive function call is analogous to traversing to a child node in a tree, while returning from a recursive function call is analogous to backtracking to a previous parent node. While recursion is overused for simple programming problems, it is well matched for problems that involve tree-like structures and backtracking. Using these ideas of tree-like structures, we wrote several algorithms for traversing, searching, and determining the depth of tree structures. We also showed that a simply connected maze has a tree-like structure, and employed recursion and backtracking to solve a maze.

• Answers to Questions: Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al

Answer: R-4.1 to R-4.5 page 180

R-4.1 Describe a recursive algorithm for finding the maximum element in a sequence, S, of n elements. What is your running time and space usage?

def find\_max(s):  
 if len(s) < 1:  
 return -1  
 if len(s) <= 1:  
 return s[0]  
 if s[0] > s[-1]:  
 return find\_max(s[:-1])  
 else:  
 return find\_max(s[1:])  
  
print(find\_max([10, 2, 3, 9, 4, 7, 3]))  
print(find\_max([1, 5, 146, 6, 8, 145, 147, 145]))

Analysis:

Find max(s) has a running time of O(n) as all operations are O(1) and the recursion will be called n times (the len of s) before returning the remaining item which will be the largest item of the sequence.

R-4.2 Draw the recursion trace for the computation of power(2,5), using the traditional function implemented in Code Fragment 4.11.

power(2, 5)

calls power(2, 4)

calls power(2, 3)

calls power(2, 2)

calls power(2, 1)

calls power(2, 0)

which returns 1

which returns 2 \* 1

which returns 2 \* (2\*1)

which returns 2 \* (2 \* 2)

which returns 2 \* ( 2\*2\*2 )

which returns 2 \* ( 2\*2\*2\*2 ) = 2 \*\* 5 = 32

R-4.3 Draw the recursion trace for the computation of power(2,18), using the repeated squaring algorithm, as implemented in Code Fragment 4.12.



return 64 \* 64 \* 2 = 8192

return 8 \* 8 = 64

return 2 \* 2 \* 2 = 8

return 1 \* 1 \* 2 = 2

return 1

Power 2, 18

Power 2, 16

Power 2, 13

Power 2, 6

Power 2, 3

Power 2, 1

Power 2, 0

R-4.4 Draw the recursion trace for the execution of function reverse(S, 0, 5) (Code Fragment 4.10) on S = [4, 3, 6, 2, 6].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 6 | 2 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | 3 | 6 | 2 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | 2 | 6 | 3 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | 2 | 6 | 3 | 4 |

R-4.5 Draw the recursion trace for the execution of function PuzzleSolve(3,S,U) (Code Fragment 4.14), where S is empty and U = {a,b,c,d}.



PuzzleSolve(2, abd, {c})

PuzzleSolve(2, bad, {c})

PuzzleSolve(2, cad, {b})

PuzzleSolve(2, dac, {b})

PuzzleSolve(2, dab, {c})

PuzzleSolve(2, cab, {d})

PuzzleSolve(2, bac, {d})

PuzzleSolve(2, abc, {d})

PuzzleSolve(2, d, {a, b, c})

PuzzleSolve(2, c, {a, b, d})

PuzzleSolve(2, b, {a, c, d})

PuzzleSolve(2, a, {b, c, d})

PuzzleSolve(3, (), {a, b, c})

2. InLab

• Write your Objectives (you can have your own objectives)

To know how to implement Recursion.

To have a knowledge of the usage of Recursion.

To apply recursion in coding world.

Implementation of Recursion

a. Write a short recursive Python function that finds the minimum and maximum values in a sequence without using any loops.

def findMinRec(A, n):  
 if (n == 1):  
 return A[0]  
 return min(A[n - 1], findMinRec(A, n - 1))  
def findMaxRec(A, n):  
 if (n == 1):  
 return A[0]  
 return max(A[n - 1], findMaxRec(A, n - 1))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1, 4, 45, 6, -50, 10, 2]  
 n = len(A)  
 print(findMinRec(A, n))  
 print(findMaxRec(A, n))

b. Give a recursive algorithm to compute the product of two positive integers, m and n, using only addition and subtraction.

def sum(x, y):  
 if (y == 0):  
 return x;  
 else:  
 return (1 + sum(x, y - 1));  
  
x = int(input("Enter number first number: "))  
y = int(input("Enter number second number: "))  
print("Sum of two numbers are: ", sum(x, y))  
  
def sub(x, y):  
 if y == 0:  
 return x  
 return sub(x - 1, y - 1)  
  
x = int(input("Enter some first number = "))  
y = int(input("Enter some second number = "))  
print("Sub of two numbers are: ", sub(x, y))

c. Write a short recursive Python function that takes a character string s and outputs its reverse. For example, the reverse of pots&pans would be snap&stop

a="pots&pans"

b=a.split("&")

def reverse(word):

if not word:

return ""

return reverse(word[1:]) + word[0]

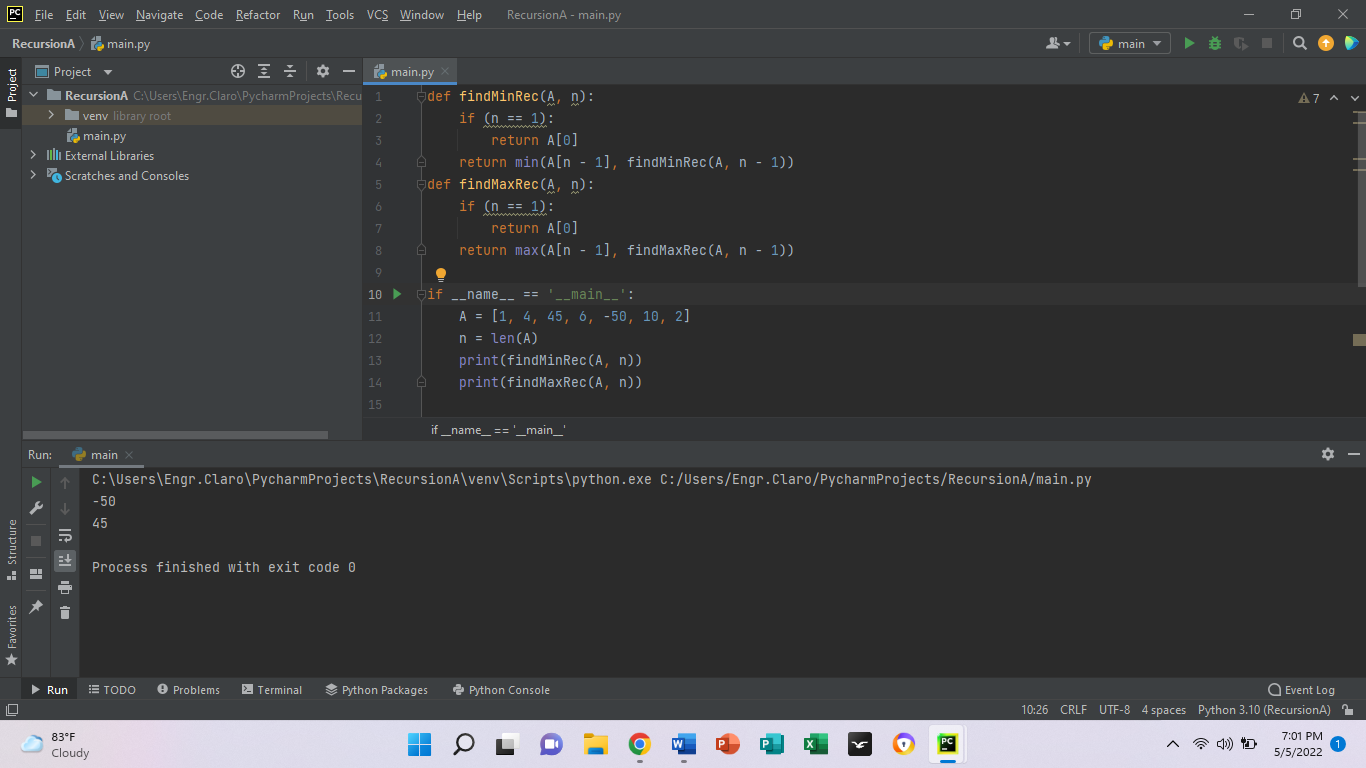
result = reverse(b[1]) + "&" + reverse(b[0])

print(result)

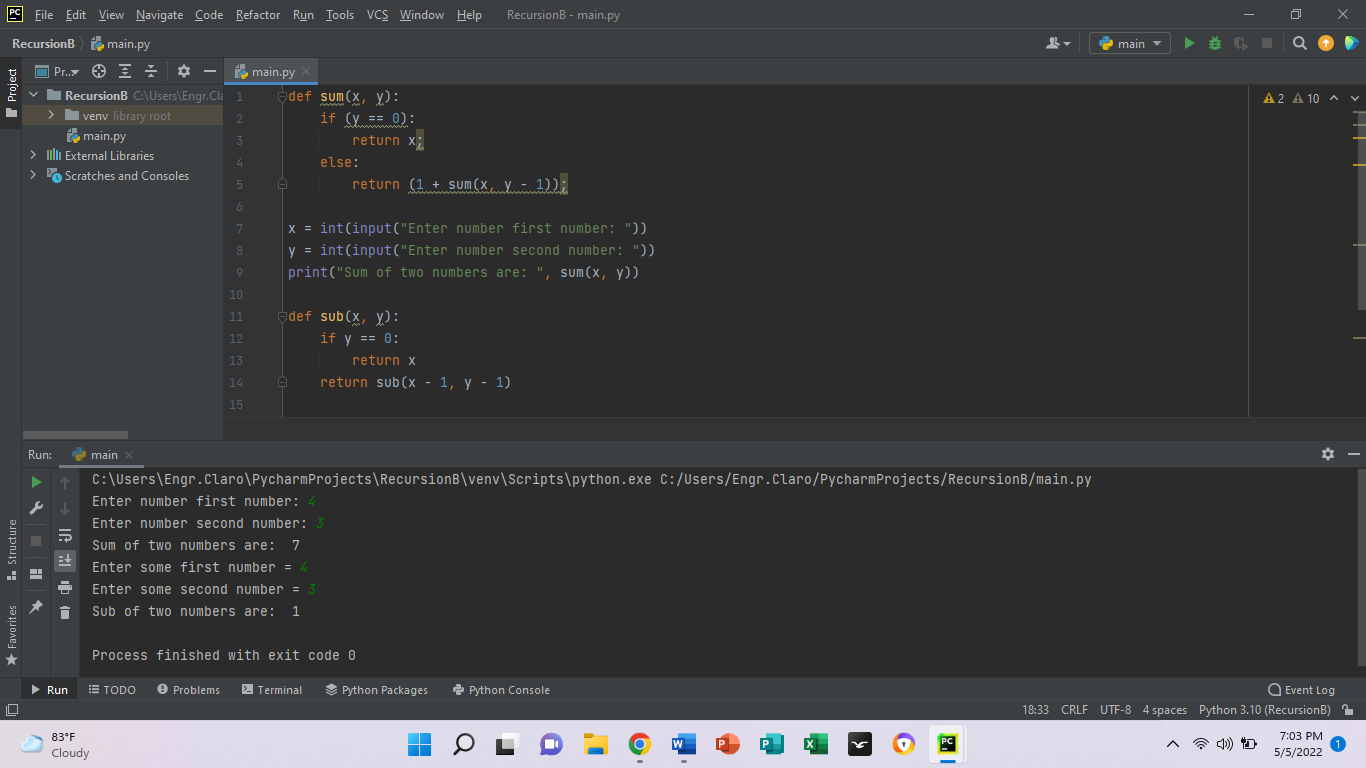
• Steps performed with screenshots of tools used, sample run with DISCUSSIONS (DONT copy and paste from the e-book).

• Edit your figures (screenshot), highlight by putting a box, give a figure number and brief description.

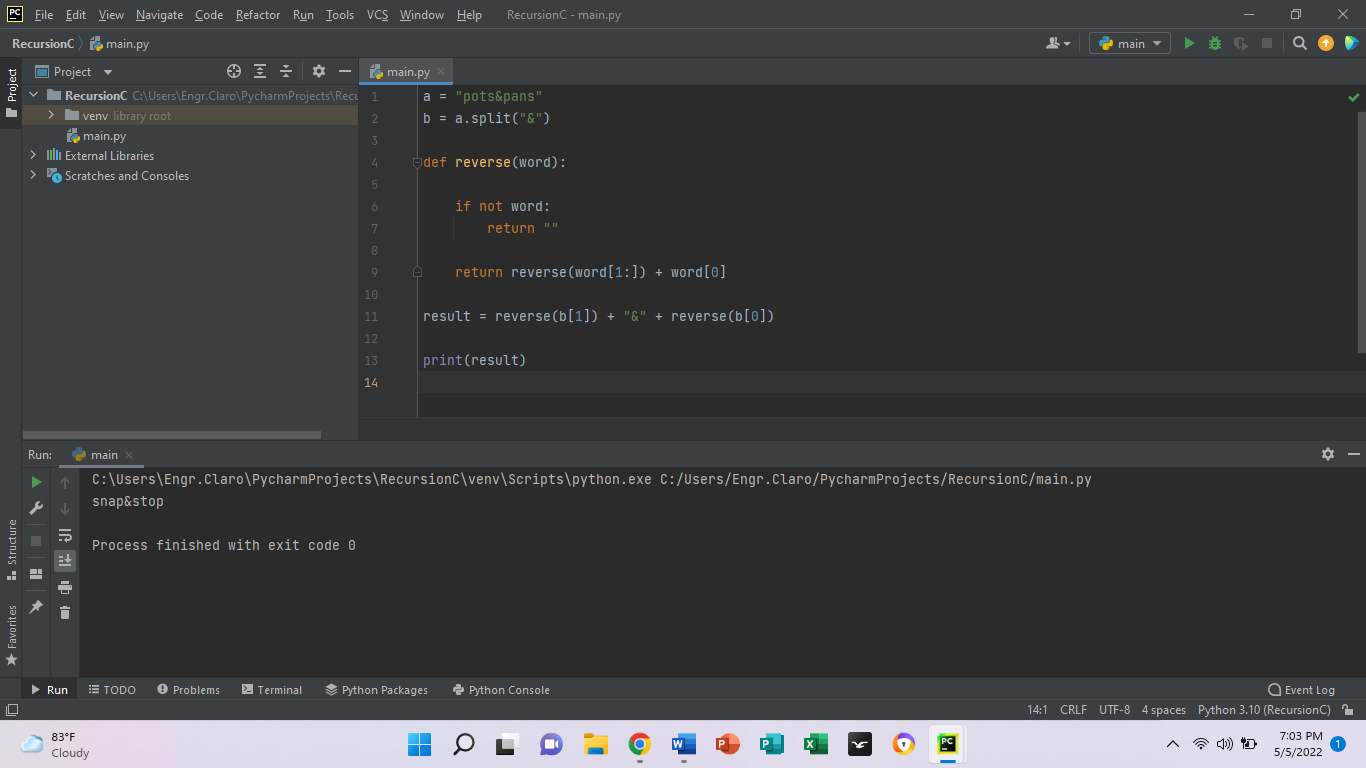
a.



b.



c.



Write your own Observation and Conclusion from what you have executed/read.

Recursion is all about is a programming technique using function or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first. Recursion is a process by which a function calls itself directly or indirectly. The corresponding function is called as recursive function.

3. PostLab

•Project

Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al: Chapter 4 Projects page 182 Do Projects P-4.23, P-4.26

P-4.23 Implement a recursive function with signature find (path, filename) that reports all entries of the file system rooted at the given path having the given file name.

import os

file\_name = "C:\Users\Engr.Claro\Documents\CLARO\_SD\_LABACT3.docx"

file\_stats = os.stat(file\_name)

print(file\_stats)

print(f'File Size in Bytes is {file\_stats.st\_size}')

P-4.24 Write a program for solving summation puzzles by enumerating and testing all possible configurations. Using your program, solve the three puzzles given in Section 4.4.3.

lst = []

num = int(input("Enter the size of the array: "))

print("Enter array elements: ")

for n in range(num):

numbers = int(input())

lst.append(numbers)

print("Sum:", sum(lst))

P-4.25 Provide a nonrecursive implementation of the draw interval function for the English ruler project of Section 4.1.2. There should be precisely 2c−1 lines of output if c represents the length of the center tick. If incrementing a counter from 0 to 2c −2, the number of dashes for each tick line should be exactly one more than the number of consecutive 1’s at the end of the binary representation of the counter.



3:4

2:3

1:2

0:1

0:2

0:4

2:4

5:6

4:5

6:7

7:8

6:8

4:6

4:8

0:8

P-4.26 Write a program that can solve instances of the Tower of Hanoi problem (from Exercise C-4.14).

def TowerOfHanoi(n , from\_rod, to\_rod, aux\_rod):

    if n == 0:

        return

    TowerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod)

    print("Move disk",n,"from rod",from\_rod,"to rod",to\_rod)

    TowerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod)

n = 4

TowerOfHanoi(n, 'A', 'C', 'B')

P-4.27 Python’s os module provides a function with signature walk(path) that is a generator yielding the tuple (dirpath, dirnames, filenames) for each subdirectory of the directory identified by string path, such that string dirpath is the full path to the subdirectory, dirnames is a list of the names of the subdirectories within dirpath, and filenames is a list of the names of non-directory entries of dirpath. For example, when visiting the cs016 subdirectory of the file system shown in Figure 4.6, the walk would yield ( /user/rt/courses/cs016 , [ homeworks , programs ], [ grades ]). Give your own implementation of such a walk function.

import os

if \_\_name\_\_ == "\_\_main\_\_":

    for (dirpath,dirnames,filenames) in os.walk('Test', topdown=true):

        print (dirpath)

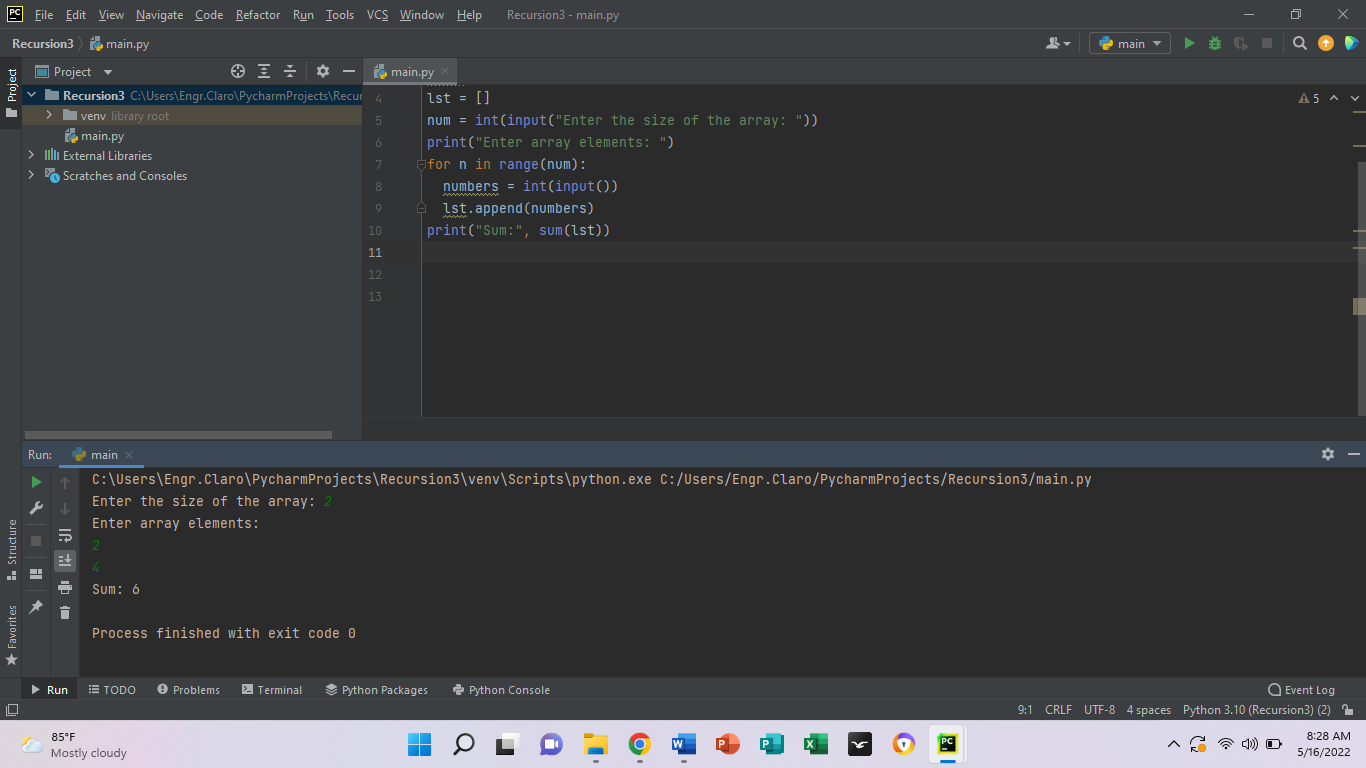
        print (dirnames)

        print (filenames)

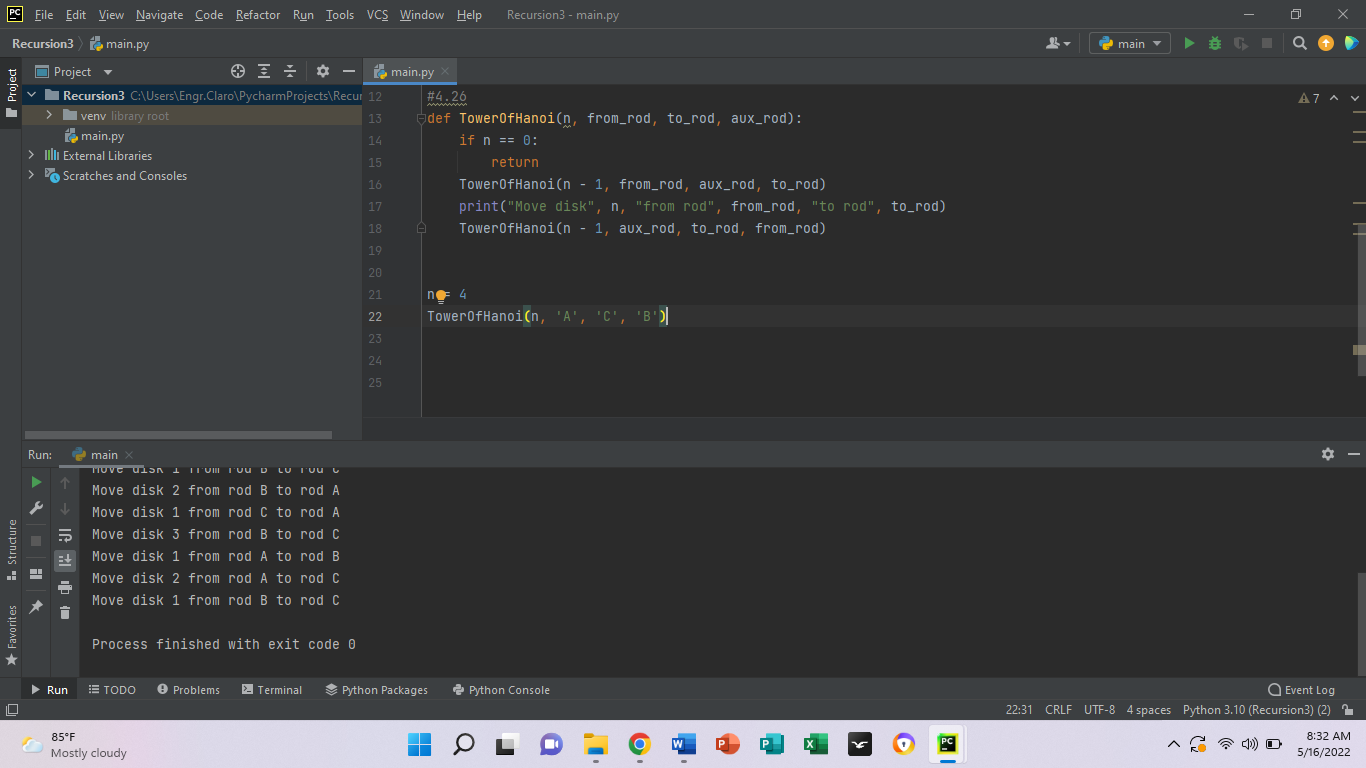
        print ('--------------------------------')

•Debugging and Sample Run

4.24



4.26



Write your own Observation and Conclusion from what you have executed/read.

In conclusion, the using of recursion is made for solving problems that can be broken down into smaller, repetitive problems. It is especially good for working on things that have many possible branches and are too complex for an iterative approach.