**Laboratory Activity 04**

**Recursion (PYTHON)**

1. PreLab

• Readings

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

**Observation**

The chapter 4 of the book Data Structures and Algorithms in Python discusses about recursion. Recursion refers to the process where a function calls itself one or more times during execution. This chapter discusses ways on how recursion could be used in Python. In recursion, techniques like a factorial function, drawing and english ruler, binary search, and file systems could be used. recursion can be done without the need for explicit loops.

(Most of the time, when the term factorial is used, it is associated with mathematics, however it can also be applied to programming.) By using recursion such as factorial function, there is no need for the use of explicit loops like the while loop or for loop. Drawing and english ruler could also be done to execute a recursion. Using ticks, a ruler can be created in Python. Another method in recursion is the binary search where it is used to locate a target value withing a sorted sequence of elements. The usage of recursion is done in sorted sequences. Another recursion technique to use is file systems. This uses directoriees linked to each other. (Each directory may contain a file and another direactory branching out.)

Despite how useful recursion is, it can still pose several problems when implemented wrongly. This can cause the recursion to perform poorly and become inefficient in the long run. One of the problems discussed in this chapter is when recursion runs amok. (An example shown where the fibonacci sequence is done poorly shows a recursion making a number of bad calls.) Another problem discussed is when a recursion turns into an infinite recursion.(From the name itself, it can be noted that this means that the recursion is done ifinitely, never reaching a base case.). This problem can affect computing resources, the CPU, and also use additional memory.

This chapter also included other examples of recursion like linear recursion where recursion starts at most one new recursive call. Binary recursion where recursion starts at two recursive calls. And multiple recursion where recursions starts at three or more recursive calls.

**Conclusion**

From this chapter, recursion can be seen as one powerful tool that programmers can use. The application of recursion is numerous and if done correctly, can be used efficiently (especially if the project is more suited in using recursion than explicit loops like while loop or for loop). However, it should be noted that a bad recursion can affect the project badly (considering a recursion could undergo an infinite recursion and resources could be wasted.) It is important to

use this tool correctly and efficiently.

* 1. 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

Observation

The book starts with discussing what recursion is. (The use of jokes to define recursion was definitely a plus and made the reading enjoyable). Just like the first reference, recursion function is defined as a function that calls itself. (The book then went back to basics by discussion functions as it emphasized the importance of understanding what functions are.) Stacks were also discussed in this book. Stacks is one of the simplest data structures in computer science that is used to store multiple values with a limitation of only adding and removing values from the top of it. Adding of values is called pushing while removing is called popping. (Compared to the first reference, this book said that recursion is not needed. That it is not required. The difference in the two references helped show that there is always going to be difference in perspectives when it comes to programming.) The inefficiency of some recursive usage such as factorials and Fibonacci is thoroughly discussed in this book and how inefficient they can be.

Another topic that was thoroughly discussed in this book is different recursive algorithms. (The way the algorithms worked are very complicated and sometimes confusing. Just like how the book said, some of those can be easily implemented with a simple loop.) These algorithms perform best with problems that are tree-like in structure and would need backtracking. Despite how complicated these recursive algorithms are, it is important to study them.

Conclusion

The conclusion is that, there is a need to study recursion, even if it may not be the best approach to some problesm, as it provides insights into different ways in tackling a problem. These would later on be helpful in future projects as references or base when solving problems. It might help in understanding how to come up with an easier solution. (The final takeaway in this book is that with recursion, practice and experience is very important. Over time, skills will improve and these will be helpful later on especially when problems involving trees and backtracking are involved)

• Questions and Answers

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

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 max(data,n):  
 if n==1:  
 return data[0]  
 else:  
 m=max(data,n-1)  
 return data[n-1] if(data[n-1]>m) else m

Because the non-recursive phase takes a fixed amount of time, a total of n recursive calls are made, resulting in an O(n) time and space complexity (n)

Diagram

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

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.

Diagram

Description automatically generated

Diagram, schematic

Description automatically generatedR-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].

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}.

Diagram

Description automatically generated

2. InLab

**Objectives**

* To be able to create simple recursive code.
* To be able to solve a basic problem using recursion.

Implementation of Recursion

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

Text

Description automatically generated

Text

Description automatically generated

Discussion:

Linear search Strategy is used. A linear search is a simple approach. The linear search determines the least and greatest values in a sequence in O time (n).

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

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

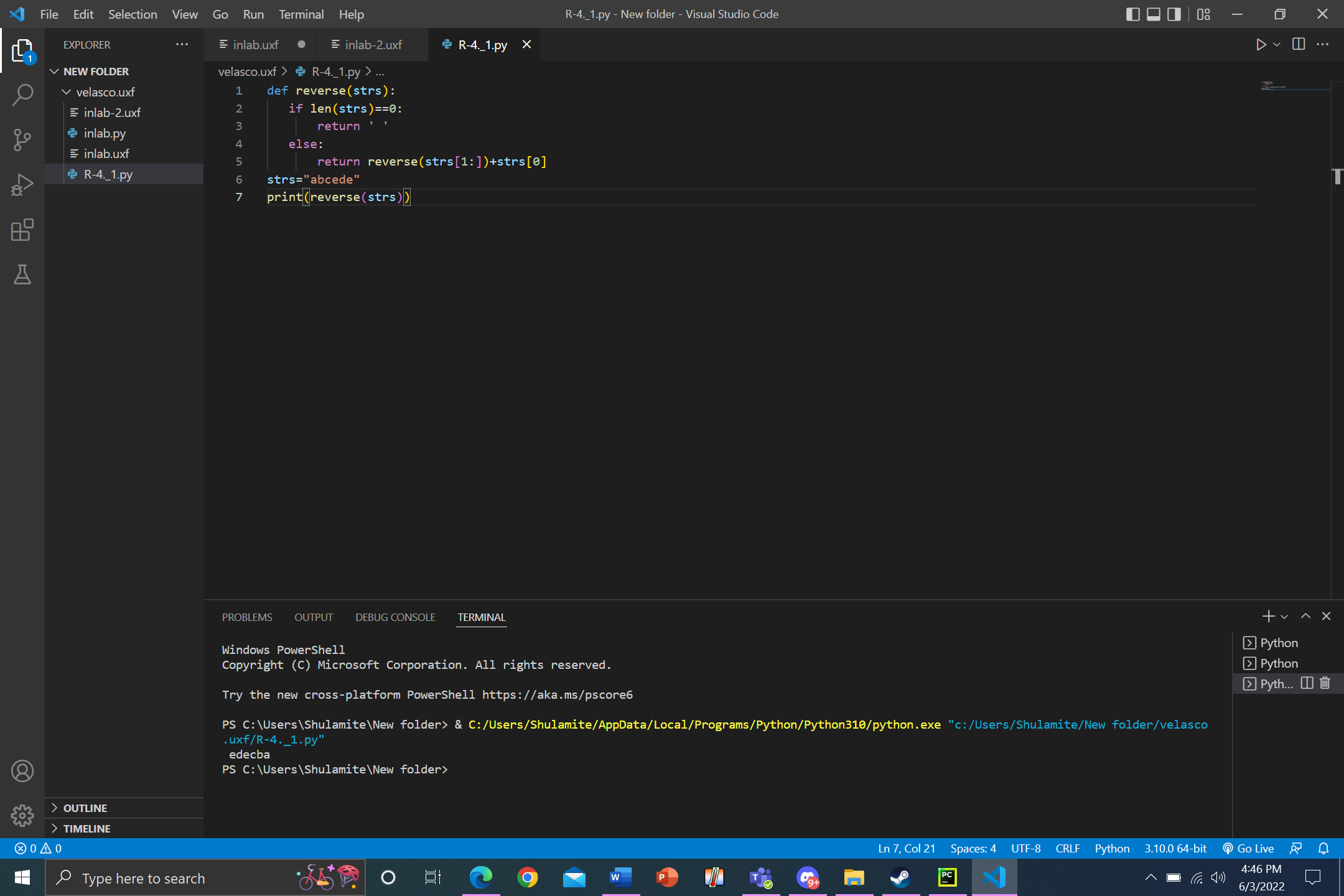
Discussion:

The solution for the current input is returned using a recursive procedure that calls itself with smaller input values and performs fundamental operations on the returned value of the smaller input, computing the product of two positive integers, m and n, using only addition and subtraction.

1. 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 screenshot of a computer

Description automatically generated



Discussion:

The function is written in such a way that each invocation of the body results in only one new recursive call; it accepts a character string s and returns its inverse. The first and last items in a series can be swapped to reverse the sequence, according to the practice of referring to this procedure as reverse for the first time (strs).

Observation

Recursion is a lot more complicated than using traditional loops. Thus, there might be confusion when trying to implement it however it has it own uses.

Conclusion

Different ways are used for different problems. Just because loops such as while loop are easier doesn’t mean it is appropriate to use. Likewise, recursion is not applicable to other problems.

**3. PostLab**

* Project

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

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.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Discussion:

In this problem, the listdir() method is used to extract a list of all files and directories in the given directory. A list of files and folders in the current working directory will be returned if no directory is given. The path () option is the directory's path. This function returns a list of all the files and folders found in the given path. The return type of this method is list.

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

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Discussion:

Three rods and n disks make up the Tower of Hanoi, a math puzzle. Although only one disk can be transported at a time, the goal of the task is to relocate the complete stack to another rod. Each move involves stacking the highest disk from one stack on top of another; a disk can only be moved if it is the topmost disk on the stack. A larger disk cannot be stacked on top of a smaller disk.

Observation

When employing recursion, the problem is not unduly complicated. When using recursive functions, it has advantages such as making your code look clean and appealing. A complex task can be broken down into smaller sub-problems via recursion.

Conclusion

To summarize, recursion has drawbacks, such as using a lot of memory and time through recursive calls, making it costly to use. Recursion, on the other hand, provides a clear and simple method of writing code. Tree traversals and the Tower of Hanoi are examples of inherently recursive problems.