Lab 4.0
Functions

Z. Hutchinson
zachary.s.hutchinson@maine.edu

October 3, 2021

Goals

The goal of this lab is to practice using Python 3. Specifically, you will practice:

- Functions

Instructions

All work is due at the **end of your lab** and must be submitted to Brightspace in the proper place. Unless otherwise instructed, submissions must be python files (e.g. files that end with *.py*). Any other format, even if it is plain text, will **not** be graded. Messy or otherwise unreadable code will lose points. Lab submissions can be all in the same file, but please label with comments to which task code belongs. IMPORTANT: Any code that is commented out will not be graded. **RUN YOUR CODE TO MAKE SURE IT WORKS!!!**

Task 1 - Simple Functions

Convert the following to Python code. IMPORTANT: You must include test code for each function.

A Write a function named, *PrintMyName*, which when called prints your name. It does not return anything. It does not have any parameters.

B Write a function named, *PrintAName*. This function has one parameter, *name*. The function prints the value stored in *name*. The function does not return anything.

C Write a function named, *GetName*. This function does not have any parameters. This function should ask the user for their for their full name. The function should return what the user enters.

D Write a function named, *Multiply*. This function has two parameters, *numA* and *numB*. The function should return the product of *numA* and *numB*.

Task 2 - Advanced Functions

Convert the following to code. Just like the previous section, you should provide tests for each of your functions. Preferably, more than one.

A Write a function called *IsNegative*. This function has one parameter, *num*, which it expects to be a numeric type. If the value in *num* is less than zero, the function returns True. Otherwise, it returns False.

B Write a function called *MultiplyList*. This function has two parameters: *alist* and *num*. It expects *alist* to be a list of numeric types and *num* to be a single number. This function replaces each value in the list with the value scaled by *num*. For example, if the first value in *alist* was 2 and *num* was equal to 1.5, the new first value in the list after the function completes would be 3. The function does not return anything.

C Write a function called *LoopOnPositive*. This function has no parameters. It contains a sentinel loop which asks the user for a number. The loop should stop if the user types a negative number. To check for negative numbers, you must use your function from Part A. In each iteration of the loop, print the square root of the number the user typed in before asking for a new number.

Task 3 - Reading Code

Read the following code and try to understand what it does. Suggest better names for each function based on your understanding of the code.

```
# alist: list
# indexA: a valid index for alist
# indexB: a valid index for alist
def FunctionA(alist, indexA, indexB):
    tmp = alist[indexA]
    alist[indexA] = alist[indexB]
    alist[indexB] = tmp

# alist: list
def FunctionB(alist):
    start = 0
    end = len(alist)-1
    while start < end:
        FunctionA(alist,start,end)
        start +=1
        end -=1

def main():
    mylist1 = [
        0,1,2,3,4,5,
        6,7,8,9,10,11
    ]
    mylist2 = [
        'Bangor',
        'Brewer',
        'Orono',
        'Portland',
        'Augusta'
    ]

    FunctionB(mylist1)
    FunctionB(mylist2)

    for x in mylist1:
        print(x)

    for x in mylist2:
        print(x)

main()
```

Task 4 - While loop Challenge

For this task you need a working version of Lab3's Task 1 Part C. Copy your working solution. The goal is to alter it in the following way. The user has a second option in the inner loop. If they type 'exit', it will quit both loops without further prompts for input. This means, if the user types 'exit', both inner and outer loops will end. This can be accomplished in several ways. They can still type 'quit' to stop just the inner loop. And of course, 'q' to end the outer loop. In other words, in spite of the 'exit' option, the code should still conform to the original task.