Tutorial 4

Problem 1.

Page 72, Section 8.3 of our online textbook:

Write a function that takes a string as a parameter and displays the letters backward, one per line. Then, write another function that takes a string as a parameter and returns the string reversed (without printing it).

Problem 2.

Consider the Bubble Sort algorithm below:

```
for end in range(len(data) - 1, 0, -1):
   for swapIndex in range(0, end, 1):
      if data[swapIndex] < data[swapIndex + 1]:
         # we swap:
         tempVar = data[swapIndex +1]
         data[swapIndex +1] = data[swapIndex]
         data[swapIndex] = tempVar
```

Hand trace the algorithm while we are sorting the following data list in a **decreasing** sort order:

4 9 6 3 1 8 2 0 5 7

As we are answering this question, let's show our work. This is to say, what did the list looks like before and after each iteration of the algorithm.

Problem 3.

Write a recursive function `printPattern(seed, depth)` such that when we call it as follows: `printPattern('a', 8)`, it produces the following output:

```
aaaaaaaa
bbbbbbb
cccccc
ddddd
eeee
fff
gg
h
h
gg
fff
eeee
ddddd
cccccc
bbbbbbb
aaaaaaaa
```

Hint: Investigate the built-in functions `ord('a')`, which produces 97, and `chr(97)` which produces `'a'`. Perhaps, we can combine them.

Why is the first parameter called '`seed`'? Why is the second parameter called '`depth`'?

Problem 4.

Exercise 8.3 from our online textbook:

A string slice can take a third index that specifies the "step size"; that is, the number of spaces between successive characters. A step size of 2 means every other character; a step size of 3 means every third, etc.

```
>>> fruit = 'banana'
>>> fruit[0:5:2]
'bnn'
```

A step size of -1 goes through the word backwards, so the slice [: : -1] generates a reversed string. Use this to write a loop-free (i.e., no loop) version of the function `is_palindrome( )`.

Would this function take parameter(s) and if so, how many? Would this function return a value and if so, what?

Problem 5.

Have a look at the Python program below:

```
variable1 = 1
variable2 = 2

def function1():
    variable3 = 3
    variable4 = variable3
    return variable4

def function2(parameter1):
    variable5 = parameter1
    variable6 = 6
    return variable5

def function3(parameter2, parameter3):
    variable7 = parameter2
    variable8 = parameter3
    return variable9

# Main part of program
variable10 = function2(variable3)
print(function3(variable5, variable2))
variable1 = function1()
```

a. What is wrong with this program?
b. In general, what is the scope of a variable?
c. In general, what is the scope of a parameter?
d. In general, what is the relationship between variable scope and the stack frame of a function?
e. What is the scope of each of the 10 variables in the Python program?
f. What is the scope of each of the 3 parameters in the Python program?

Problem 6. The keyword **`None`**

The list methods `append( )` and `insert( )` modify the list that calls these methods. For example, the list `aList` is modified by the following statement:

```
aList.insert(2,26)
```

So, if `aList` was `[1, 2, 3, 4]`, it then becomes `[1, 2, 26, 3, 4]`. Hence, we do not need to reassign the result of these methods back to the list. This is to say that we do now have to do the following:

```
alist = aList.insert(2,26)
```

Actually, the above statement will destroy our `aList`. To see why that is, execute the following Python code fragment and see what it does:

Python Code Fragment:

```
aList = [1,2,3,4,5]
aList = aList.append(8)
print(aList)
```

List methods such as `insert( )`, `append( )`, and `remove( )` modify the list that called them, i.e., the list on the left hand side of the "**.**" (access operator). These methods do not return anything (such as a list, a number, etc…). When Python functions and methods do not return any value, **`None`** is returned. In the above Python code fragment, the `aList` is assigned the value **`None`** because `append( )` does not return a value. Hence, **`None`** is printed on the computer monitor screen when the last statement, i.e., the `print` statement `print(aList)` is executed.