**Chapter 6 – Week 9 – Exercises**

Exercises #1 – page 171

1. **Anne complains that defining functions to use in her programs is a lot of extra work. She says she can finish her programs much more quickly if she just writes them using the basic operations and control statements. State three reasons why her view is shortsighted.**

This view is shortsighted because:

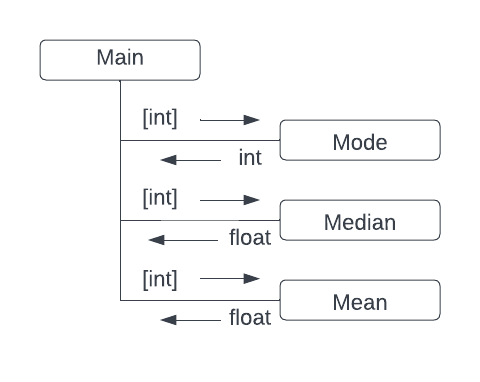
* 1. It shows a lack of understanding about code reuse. Functions are much easier to reuse than entire programs.
  2. It shows a lack of appreciation of code maintenance. Functions, used properly, make code simpler and simpler code is easier to maintain.
  3. It makes the assumption that only one programmer will be working on the project. Every source control tool I have used allows “ownership” of code by file. Since a single program will be in a single file and only one programmer can change a single file, it prevents multiple programmers from working together. This limits the size the project.

1. **Explain how an algorithm solves a general class of problems and how a function definition can support this property of an algorithm.**

A general algorithm is a series of steps that allow a given problem to be solved given an initial state. A function allows the initial state to be provided using parameters.

Exercises #2 – page 176

1. **Draw a structure chart for one of the solutions to the programming projects of Chapters 4 and 5 The program should include at least two function definitions other than the main function.**



Structure chart for chapter 5 problem #1.

1. **Describe the processes of top-down design and stepwise refinement. Where does the design start, and how does it proceed?**

Top-down design is the process of taking a global view of the problem and breaking the problem up into a set of sub-problems.

Often, each sub-problem is defined in a function. This sub-problem may need to be broken into its own set of sub-problems.

As each sub-problem is defined and solved, it is one step closer to the total solution. Each function and solved sub-problem is considered a step toward the solution.

The process of taking “step” after “step” toward the global solution is known as **stepwise refinement**.

Exercises #3 – page 182

1. **In what way is a recursive design different from top-down design?**

Top-down design divides a problem domain into smaller problems. A recursive design divides a problem domain into a smaller instance of the same design.

1. **The factorial of a positive integer n, fact(n), is defined recursively as follows:**

fact(n) = 1, when n = 1

fact(n) = n \* fact(n-1), otherwise

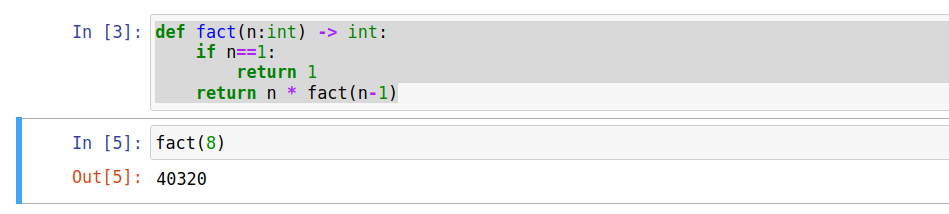
**Define a recursive function fact that returns the factorial of a given positive integer.**

def fact(n:int) -> int:

if n==1:

return 1

return n \* fact(n-1)



1. **Describe the costs and benefits of defining and using a recursive function.**

A recursive function is often a trade off between code simplicity and space usage where the program stack grows in size.

Thus, the cost is memory and the benefit is simpler code.

1. **Explain what happens when the following recursive function is called with the value 4 as an argument:**

def example(n):

if n > 0:

print (n)

example(n-1)

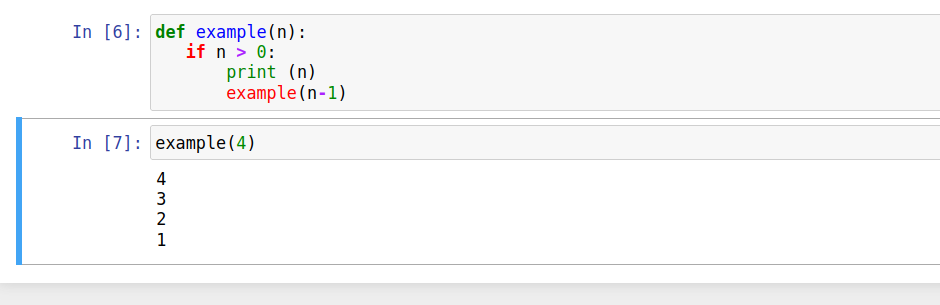
example(4) will print

4

3

2

1



1. **Explain what happens when the following recursive function is called with the values “hello” and 0 as arguments:**

def example(n):

if n > 0:

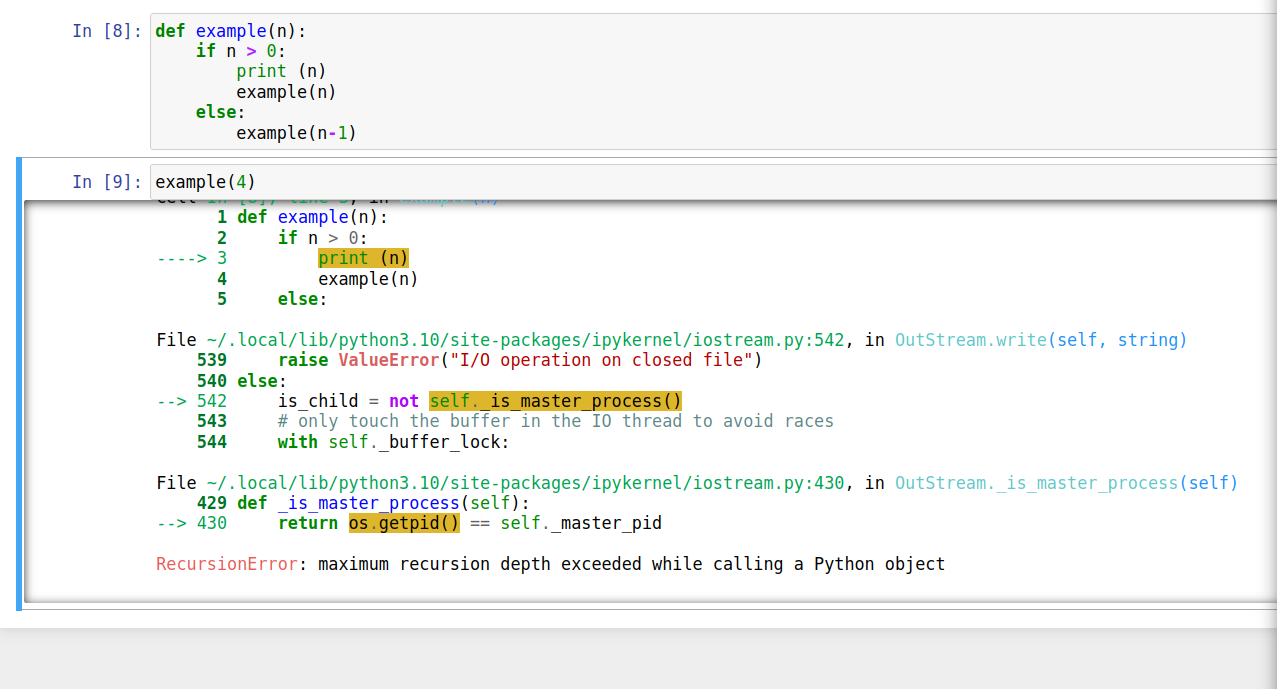
print (n)

example(n)

else:

example(n-1)

example(4) will print the value “4” on line after line until a RecursionError is received.



1. **Explain what happens when the following recursive function is called with the values “hello” and 0 as arguments:**

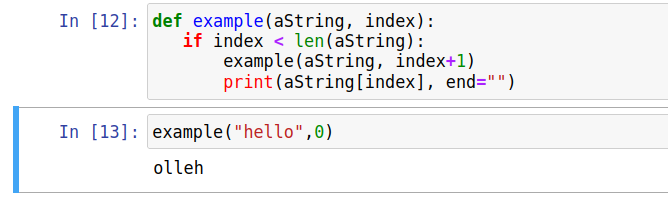
def example(aString, index):

if index < len(aString):

example(aString, index+1)

print(aString[index], end=””)

example(“hello”,0) will print hello backwards (“olleh”)



1. **Explain what happens when the following recursive function is called with the values “hello” and 0 as arguments:**

def example(aString, index):

if index == len(aString):

return “”

else:

return aString[index] + example(aString, index+1)

calling this code with example(“hello”,0) will return the value supplied to the function (“hello”).

Exercises #4 – page 194

1. **Where are module variables, parameters, and temporary variables introduced and initialized in a program?**

Module variables are introduced and initialized at the level of the module. They receive their values at the moment they are introduced. The will be usually be found starting at the first character of a line and will be found outside of all containers including functions. They can be found using the “dir” command.

Parameters are introduced and initialized in the definition of a function. A function definition starts with the keyword “def” followed by the function name followed by the parameter list. The parameter list is enclosed in parenthesis () and consists of the parameter names and optionally their data type.

Parameters are initialized either in the function definition (in the case of default values) or in the function call when the parameter values are specified either by name or by position.

Temporary variables are are introduced and initialized in the body of a function. They receive their values at the moment they are introduced.

1. **What is the scope of a variable? Give an example.**

In Python, the scope of a variable is the area of the program text in which the name refers to a given value.

For example, the scope of a module variable begins at the location of it definition and continues until the end of the file. A temporary variable defined in a function will generally be in scope from the time of its definition until the end of the function.

Another example is seen here. The scope of the variable scope\_example is shown in bold:

is\_blue = True

if is\_blue:

**scope\_example = 64**

**test = scope\_example \* 37**

else:

print(“Scope does not extend to here.”

1. **What is the lifetime of a variable? Give an example.**

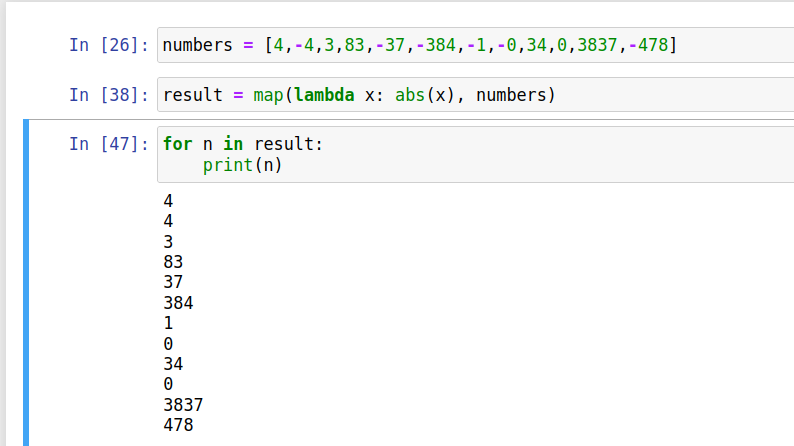
The lifetime of a variable is the period of time during program execution when the variable has memory storage associated with it.

While **scope** makes reference where in the **code** a variable name can be used, **lifetime** refers to when the variable can be accessed during the **execution of the program**.

Exercises #5 – page 199-200

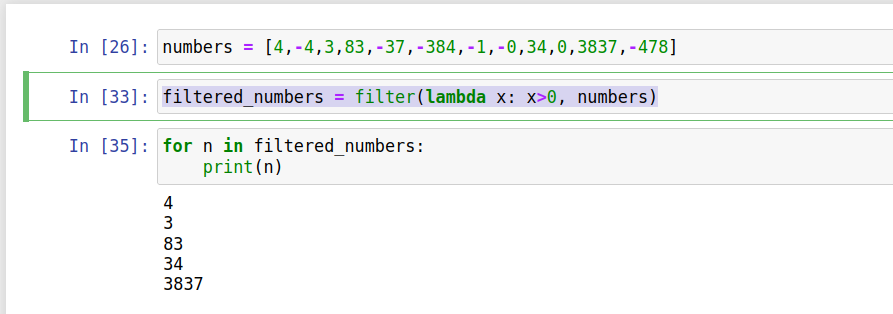
1. **Write the code for a mapping that generates a list of the absolute values of the numbers in a list named numbers.**

map(lambda x: abs(x), numbers)

****

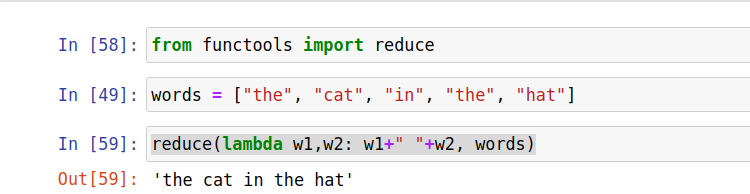
1. **Write the code for a filtering that generates a list of the positive numbers in a list named numbers. You should use a lambda to create the auxiliary function.**

filtered\_numbers = filter(lambda x: x>0, numbers)



1. **Write the code for a reducing that creates a single string from a list of strings named words.**

reduce(lambda w1,w2: w1+" "+w2, words)

****

1. **Modify the summation function presented in Section 6.2 so that it includes default arguments for a step value and a function. The step value is used to move to the next value in the range. The function is applied to each number visited, and the function’s returned value is added to the running total. The default step value is 1, and the default function is lambda that returns its argument (essentially an identity function). An example call of this function is summation (1,100,2, math.sqrt) which returns the sum of the square roots of every other number between 1 and 100. The function can also be called as usual, with just the bounds of the range.**

def summation(lower,upper,step=1,fn=lambda x: x):

""" Returns the sum of the numbers from lower

to upper.

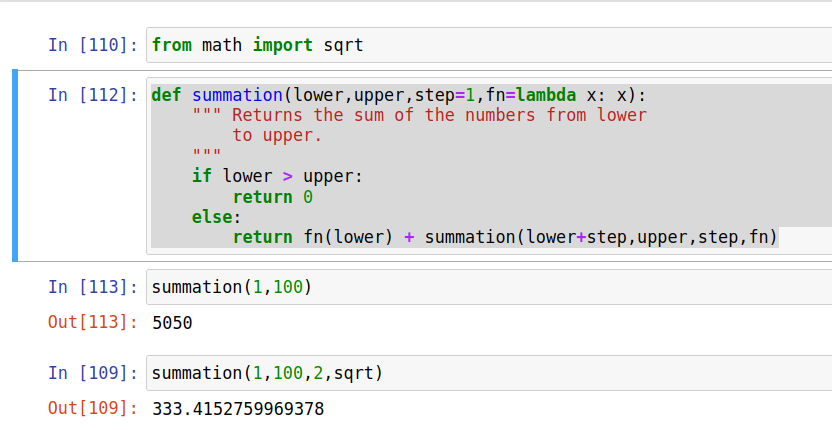
"""

if lower > upper:

return 0

else:

return fn(lower) + summation(lower+step,upper,step,fn)

****

1. **Three versions of summation function have been presented in this chapter. One uses a loop, one uses recursion, and one uses the reduce function. Discuss the costs and benefits of each version in terms of programmer time and computational resources required.**

Due to the simplicity of the code in these example, the programmer and computational resources are similar in all 3 cases. The all are fairly minimal.

In a more general case, writing a loop is the most common of the three for new programmers. However, for an experienced programmer it is likely the most complicated. The loop termination condition must be considered carefully.

On the other hand, the reduce function handles the termination condition within the function call. This removes the most complicated portion of the problem.

Recursion is in the middle. It takes less contemplation than a loop, but more than a previously written function.

When it comes to the amount of memory and computation time required, the recursive function is likely to require the longest. This is due to the need to add a stack frame for each iteration of the “loop” in question. (As an aside, a functional programming languages generally do not need to add these stack frames if the programmer makes the recursive call the last action in the function.) This means the loop will require less memory and time that the recursive function.

The reduce function’s performance depends on its implementation. However, since it is in a Python standard library, it is likely to perform at near maximum effeciency.