Week 6, Lecture 1

Please read Chapter 6 of Zelle, on functions.

Today's topic is "functions".

Now you already know how to define functions, call functions, and have functions either

(a) run and do some work but return no value/values, or
(b) return one or more values.

You've seen that Python (the shell) will run any expression or sequence of expressions you give to it,
directly to the shell or via a file in which you type the expressions. You can do elaborate computations
that way, if you want.

It is, however, more convenient to use functions — because functions can be small or large units of computation
that take in arguments (parameters), and return values, and you can call these from anywhere in your execution sequence.

Each function can have parameters, and yes, even main() can have parameters, though we'll see that later.

Function main() acts as a manager. It is the first function to run, and it will also be the last to run, unless

some function makes a “sys” call for the program to exit() the CPU.

Every other function you write is either called by main() or by some other function.

When you use variables in functions you must be aware of what “kind” of variables they are. That is,
you must be aware of the SCOPE of each variable. The SCOPE of a variable has to do with whether
A variable inside a function is or is not accessible (i.e., can some other function see/touch it?) to
some other function.

So we have to know which variables are “local” to a function, which variables are parameters to a
function (all parameters are local to a function) etc. Local variables are also called “automatic”
Variables because they “come and go”, meaning they exist only as long as a function runs — not
before and not after.

You can make a variable “global” so that all functions in the module in which you declare this global
will have access to this variable. You may be tempted to use such a variable in a crunch, because it
helps you avoid defining another parameter. But this is dangerous in general. Because the more easily

accessible a variable is to arbitrary functions, the more likely it is that things can go wrong.

Later when learn about “classes” you’ll find that “static” variables do this work for you.

Thus far, when we passed parameters to functions, the functions used those parameters, computed some results and then returned some value(s) to the calling function. This is one way of doing things.

But sometimes we want to pass some information (i.e., parameters) to a function and need the function to actually change the values of these parameters inside the function after doing some computation.
In this case the function need not return anything to the caller. It’s work is reflected in how it changed the parameter values. The important point is this: when the function returns, the calling function will find that called function updated the variables (passed as parameters) inside the called function.
Unlike C, Python does this without the user having to work with “pointers”.

Finally, we build a simple model of coin tossing. We make a sequence of n tosses of either a
fair coin (probability of Heads is 0.5), or an unfair coin (probability of Heads is not 0.5).

Experiment with this code and the probabilities, to see of the results make sense to you.
