Chapter 4 Notes

Functions

Function is defined in Python with the keyword “def”. The sequence of names within the parentheses following the function name are formal parameters of the function. When the function is used (invoked), the parameters are bound (as in assignment statement) to the actual (real) parameters (arguments) of the function call.

There is a special statement, **return**, that can be used only within the body of a function. A function invocation is an expression and has a value. That value is the value returned by the invoked function.

When a function is called:

* The actual parameters are evaluated and assigned into formal parameters in the function.
* The point of execution moves from the main program into the function’s statements.
* The body of the function is executed until either a return statement is encountered, in which case the value of the expression following the return keyword becomes the value of the function call, or there are no more statement to execute, in which case the function returns the value None.
* The point of execution is transferred back to the main program.

Parameters allows programmers to write code that can manipulate not specific objects, but instead whatever objects the caller of the function choose to use as actual parameters. This is called Lambda Abstraction.

Keyword arguments and Positional arguments.

**Positional argument** is the method in which the first formal parameter is bound to the first actual parameter and the second formal to the second actual, etc.

**Keyword arguments** is the method in which formals are bound to actuals using the name of the formal parameter. For example function(first\_par, second\_par). For this function we can invoke them as function(second\_par = “variable1”, first\_par = “variable2”). Here, you bound actuals to formals by formals’ names.

Though the keyword arguments can appear in any order in the list of actual parameters, it is not legal to follow a keyword argument with a non-keyword argument.

Keyword arguments are commonly used in conjunction with **default parameter** values. Default parameters are parameters that were defined in the formal parameter at the definition step. It allows programmers to call a function with fewer than the than the specified number of arguments.

Scoping

It is important to note that though the actual parameters have the same name, they are not the same variable. Each function defines a new name space, or scope.

Specification

The text between the triple quotation marks is called a docstring in Python. By convention, programmers write docstrings to provide specifications of functions. These docstrings can be accessed using the built-in function “**help()**”

A specification of a function defines a contract between the implementer of a function and those who will be writing programs that use the function (clients). This contract includes:

* **Assumptions**: These describe conditions that must be met by clients of the function. Most often, they describe the constraints on the actual parameters.
* **Guarantees**: These describe conditions that must be met by the function, provided that it has been called in a way that satisfied the assumptions.

Abstraction and Decomposition

Functions are a way of creating computational elements that we can think of as primitive. They also facilitate this by providing decomposition and abstraction.

* Decomposition creates structure. It allows us to break a program into parts that are reasonably self-contained, and that may be reused in different settings.
* Abstraction hides detail. It allows us to use a piece of code as if it were a black box. The essence of abstraction is preserving information that is relevant in each context and forgetting information that is irrelevant in that context.

Recursion

Recursion is a method is problem solving that allows its user to break down a problem into smaller parts and solve them. In general, a recursive definition is made up of two parts.

One, there is at least one base case that directly specifies the result for a special case; and Two, there is at least one recursive (inductive) case that defines the answer in terms of the answer to the question on some other input, typically a simpler version of the same problem.

The problem-solving principle is to conquer a hard problem by breaking it into a set of subproblems with the properties that:

* The subproblems are easier to solve than the original problem, and
* Solutions of the subproblems can be combined to solve the original problem.

**Global variables** are variables that can be accessed by all scopes within a main program.

Modules

It is typically more convenient to store different parts of a program in different files for large programs. Python modules allow us to easily construct a program from code in multiple files. A module is a .py file containing Python definition and statements. A program can get access to a module through an import statement.

To invoke certain function or variable from a module we use the dot notation as

* module\_name.object\_name
* module\_name.function\_name()

Files

Every computer system uses files to save things from one computation to the next. Python achieves operating-system independence by accessing files through something called a file handle. With Python, you can access a file through file handle using the code:

* Handlesname = **open(‘file\_name’, ‘r’)** (r stands for read)
* Handlesname = **open(‘file\_name’, ‘w’)** (w stands for write, which means that we can now rewrite the whole file (overwrite))
  + Handlesname**.write(“string” + ‘\n’)** the ‘\n’ is optional if you want to goes down a line.
* **Handlesname.close()**, this comes at the end when you have finished using that file.
* Handlesname = open**(‘file\_name’, ‘a’)** (a stands for append, which means that we can add more information into the file without rewrite the whole thing).

