CHAPTER

EIGHT

LECTURE 5 — PYTHON FUNCTIONS

8.1 Reading

Most of this is covered late in Chapter 3 of *Practical Programming*.

8.2 Why Functions?

- The purpose of today's class is to introduce the basics of writing and running Python functions.
- Recall Lab 1 for computing disk size: We had to repeat the same computation three times for different inputs.
  - Repeating code is painful.
  - It is also hard to distinguish between the same code repeated three times and three different computations.
  - It is easy to find a mistake in one copy of a section of code and forget to fix it in the other copies.
- Learn a programmer's motto: DRY – don't repeat yourself.
  - Define it once and use it multiple times.
- Functions are extremely useful for writing complex programs:
  - They divide complex operations into a combination of simpler steps.
  - They make programs easier to read and debug by abstracting out frequently repeated code.

8.3 Functions

- As we have learned, a function
  - Takes as input one or more arguments.
  - Computes a new value, a string or a number.
  - Returns the value, so that it can be assigned to a variable or output.
- Let's recall this with a built-in function:

```
>>> len('RPI Puckman')
11
```

Can you identify the input argument, the computation and the returned value?

8.4 A Function to Compute the Area of a Circle

- In mathematics, many functions are given as formulas. You might write a function to calculate the area of a circle as

$$a(r) = \pi r^2$$

- In Python, typing into a file (in the upper pane of the Wing IDE), we write

```
def area_circle(radius):
    pi = 3.14159
    area = pi * radius**2
    return area
```

- Note that the `def` is not indented and the other lines are indented four spaces.
- We add unindented code the file below the definition of `area_circle()` to execute the function and calculate the area:

```
a = area_circle(1)
print(a)
print('A circle with radius 2 has area {:.2f}'.format(area_circle(2)))
r = 75.1
a = area_circle(r)
print("A circle with radius {:.2f} has area {:.2f}".format(r,a))
```

  Note that by using examples with small values for the radius we can easily check that our function is correct.

- Important syntax includes
  - Use of the keyword `def` and the `:` to indicate the start of the function
  - Indentation for the lines after the `def` line
  - Unindented lines for code outside the function to indicate the end of the function.

8.5 What does Python do when we run this code?

1. Reads the keyword `def` and notes that a function is being defined.
   - The line that starts with `def` is called the function *header*.
2. Reads the rest of the function definition, checking its syntax.
3. Notes the end of the definition when the unindented code is reached.
4. Sees the function call inside the assignment statement

```
a = area_circle(1)
```

   at what’s known as the “top level” or “main level” of execution and
   - Jumps back up to the function
   - Assigns 1 to the parameter `radius`.
   - Runs the code inside the function using `radius` as a variable inside the function.
   - Returns the result of the calculation back to the top level and assigns the value 3.14159 to the variable `a`.

5. Repeats the process of running the function at the second `print()`, this time with the parameter value 2 and therefore a new value for `radius` inside the function.
6. Repeats the process of running the function right after we reassign `a`, this time with parameter value 75.1 taken from the variable `r`.

8.6 Flow of Control

- To re-iterate, the "flow of control" of Python here involves
  - Reading the function definition without executing the function.
  - Seeing a "call" to the function, jumping up to the start of the function and executing its code.
  - Returning the result of the function back to the place in the program that called the function and continuing the execution.
- Functions can compute many different things and return any data type Python supports.

8.7 Arguments, Parameters and Local Variables

- *Arguments* are the values 1, 2 and 75.1 in our above examples.
- These are each passed to the *parameter* called `radius` named in the function header. This parameter is used just like a variable in the function.
- The variable `pi` and `area` are *local variables* to the function (though we should probably use the `math` module for `pi` in the future).
- Neither `pi` nor `radius` nor `area` exists at the top / main level. At this level, they are ''undefined variables''. Try it out.

8.8 Exercise / Example

As some basic practice, we'll write a function to convert miles per hour to kilometers per day, and then we'll write several calls to demonstrate its use. Then we will give you time to work on the first lecture exercise.

8.9 A More Complicated Example

- We'll use the example below to illustrate two important concepts:
  1. Functions can call other functions, ones that we write ourselves or that Python provides.
  2. Functions may have multiple parameters. One argument in the function call is required for each parameter. Arguments and parameters are matched up *in order*.
     - There are ways to override this, but we will not study them yet.
- In the example we will switch to the use of `math.pi` and we will

  use this from now on.

8.10 Computing the Surface Area of A Cylinder Using Two Functions

- Here is the Python code, in file `surface_area.py`:

```
import math

def area_circle(radius):
    return math.pi * radius ** 2

def area_cylinder(radius,height):
    circle_area = area_circle(radius)
    height_area = 2 * radius * math.pi * height
    return 2*circle_area + height_area

print('The area of a circle of radius 1 is', round(area_circle(1),2))
r = 2
height = 10
print('The surface area of a cylinder with radius', r)
print('and height', height, 'is', round(area_cylinder(r,height),2))
```

- Now we've defined two functions, one of which calls the other.
- Flow of control proceeds in two different ways here:
  1. Starting at the first `print()` function call at the top level, into `area_circle()` and back.
  2. At the third `print()`
     (a) into `area_cylinder()`
     (b) into `area_circle()`
     (c) back to `area_cylinder()`, and
     (d) back to the top level (and then into `round()` and finally into `print()`).
- The Python interpreter keeps track of where it is working and where to return to when it is done with a function, even if it is back into another function.
- What are the arguments, the parameters, the local variables and the global variables?

8.11 Practice Problems

1. Write a function that computes the area of a rectangle. Then, write a second function that calls this function three times to compute the surface area of a rectangular solid.
2. Write a function that returns the middle value among three integers. (Hint: make use of `min()` and `max()`.) Write code to test this function with different inputs.

You will notice that the solution to the first problem in particular longer than the solution without using functions. While we don't often write such short functions in practice, here it is a good illustration.

8.12 More on program structure

- Let us revisit the program structure that will allow us to write readable programs.
  - First a general comment describing the program.

  - Second, all import statements.
  - Third, all function definitions.
  - Fourth, the main body of your program.
- Well structured programs are easy to read and debug. We will work hard to help you develop good habits early on.

8.13 Thinking About What You See

Why is it NOT a mistake to use the same name, for example `radius`, in different functions (and sometimes at the top level)?

8.14 Let's Make Some Mistakes

In order to check our understanding, we will play around with the code and make some mistakes on purpose

- Removing `math` from `math.pi` in one definition
- Changing the name of a function
- Switching the order of the parameters in a function call
- Making an error in our calculation
- Calling `print()` with the result of a function that does not return a value

8.15 A Final Example, Including Documentation

- We will write a function that linearly scales a test score, so that if `raw` is the "raw score" then the scaled score will be

```
scaled = a*raw + b
```

  The values of `raw`, `a` and `b` will be parameters of the function. We will also want to cap the scaled score at 100.
- In writing our function we will be careful to document the meaning of the parameters and the assumptions. You should get into the habit of doing this.

8.16 Why Functions?

Our goal in using functions is to write code that is

- Easier to think about and write
- Easier to test: we can check the correctness of `area_circle` before we test `area_cylinder`.
- Clearer for someone else to read
- Reusable in other programs

Together these define the notion of *encapsulation*, another important idea in computer science!

8.17 Summary

- Functions for encapsulation and reuse
- Function syntax
- Arguments, parameters and local variables
- Flow of control, including functions that call other functions
- You can find the code developed in this class under the class modules for Lecture 5.

8.18 Additional Practice:

The lecture exercises provide just the beginnings of what you need to do to practice with the concepts and with writing short problems. Here is some additional material for review before our next class. It goes all the way back to Lecture 2...

- **Expressions:** What type of data do they return?
- Try typing simple math formula to the Python interpreter:

```
>>> 1 + 2 * 3 / 3 * 4**2 **3 - 3 / 3*4
```

  and manually find the output. Don't be fooled by the spaces! Operator precedence is in effect. Try writing your own expressions.

- **Variables:** Do you know what are valid and invalid variable names?
- What is the difference in the output between:

```
>>> 3 + 4
>>> print(3 + 4)
>>> x = 3 + 4
>>> print(x)
>>> print(x = 3+4)      # This causes an error...
```

  Try to guess before typing it in, but make a habit of typing simple statements like this and looking at the result.

- **Assignment:** Can you trace the value of a variable after many different assignments? Don't be fooled by the name of variables. Try to do it manually:

```
>>> one = 2
>>> two = 1
>>> three = 4
>>> one += 3 * two
>>> two -= 3 * one + three
```

  By the way, make a habit of picking nice variable names. Your variables should be meaningful whenever possible both to you and to anyone else reading your code.

- **Functions:** Write the functions from class on your own using the Python interpreter. Try to do it without looking at notes. Can you do it?
  - Write a function that returns a value.
  - Write a function with no return.
  - Write a function where `return` is not the last statement in the function.

  - Call these functions by either printing their result or assigning their results to a value. Here, I'll get you started.

```
def regenerate_doctor(doctor_number):
    return doctor_number+1

def regenerate_tardis(doctor_number):
    print "Tardis is now ready for doctor number", doctor_number

def eliminate_doctor(doctor_number):
    return 0
    print "You will be eliminated doctor", doctor_number
```

  - Write functions that use the built-in functions. Make sure you memorize what they are and how they are used.

- Finally, write some functions to a file and execute them from within the file. Now, execute the file. By next class, make sure all of this is quite easy to do without consulting the course notes.