Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2014

Lab 2 - Understanding Function Definitions and Function Execution

Objectives

- To gain experience developing Python functions, using the Online Python Tutor to visualize the execution of the code, and using the shell to interactively test functions that are stored in a script.

Attendance/Demo

To receive credit for this lab, you must make reasonable progress towards completing the exercises and demonstrate the code you complete. **Also, you must submit your lab work to cuLearn by the end of the lab period**. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

Getting Started

Step 1: Create a new folder named `Lab 2`.

Step 2: Launch Wing IDE 101. Check the message Python displays in the shell window, and verify that Wing is running Python version 3.4. If another version is running, ask your instructor or a TA for help to reconfigure Wing to run Python 3.4.

Exercise 1 - Defining a Simple Function and Tracing its Execution

Step 1: Recall that the area of a disk with radius r is approximately $3.14 \times r^2$, where variable r represents any positive number. To calculate the area of a disk with radius 5, we substitute 5 for r in the formula, and evaluate the expression:

$$\begin{aligned} & 3.14 \times 5^2 \\ = \; & 3.14 \times 25 \\ = \; & 78.5 \end{aligned}$$

Performing this calculation in Python is easy. At the shell prompt (`>>>`) type this expression:

```
>>> 3.14 * 5 ** 2
```

What value is displayed when this expression is evaluated?

Step 2: Here is the definition for a Python *function* that has one argument, `r`, which is the radius of a disk. The function calculates and returns the disk's area. Type this function definition in the shell. When you type the function header (the first line), the shell recognizes that you are defining a function, and repeatedly displays the secondary shell prompt (`...`) while you type the statements in the function body.

```
>>> def area_of_disk(r):
...     area = 3.14 * r ** 2
...     return area
...     # Press the Enter key here to indicate that you have
        # finished defining the function
>>>
```

Step 3: To calculate the area of a disk with radius 5, we call this function with 5 as the argument . Type this statement:

```
>>> area_of_disk(5)
```

What value is returned by the function and displayed by the shell? Is it the same as the area that was displayed in Step 1?

Step 4: To calculate the area of a disk with a different radius, we call the function with a different argument. Predict the value that `area_of_disk` should return if it called to calculate the area of a disk of radius 10. Call the function to do this. Is the result displayed by Python what you predicted?

Step 5: Function arguments are expressions. When the function is called, the expression is evaluated and that value is bound to the corresponding parameter. Try these function calls, and note the results:

```
>>> area_of_disk(2 + 3)

>>> radius = 2 + 3
>>> area_of_disk(radius)

>>> area_of_disk(radius * 2)
```

Step 6: Often, we want to save the value returned by a function for use in subsequent calculations. We do this by binding the returned value to a variable. Try this:

```
>>> result = area_of_disk(5)


>>> result
```

What value was bound to `result`?

This experiment demonstrates that a function call represents a value: the value returned by the function. This means that function calls can appear in expressions.

Step 7: We're now going to use the Online Python Tutor (OPT) to help you understand what happens as the computer executes each line of the source code, step-by-step.

On the cuLearn page for this course, look for the link Online Python Tutor - Lab 2. Click the link. The OPT editor will open in a new window. Type this *script* in the editor:

```
def area_of_disk(r):
    area = 3.14 * r ** 2
    return area

result = area_of_disk(5)
```

Ensure that OPT is configured this way: "Python 3.3", "hide frames of exited functions", "render all objects on the heap", "hide environment parent pointers", "draw references using arrows", and "show everything". If necessary, select the correct options from the drop-down menus.

Click the Visualize Execution button. Execute the script, one statement at a time, by clicking the Forward> button. Observe the memory diagram as the script is executed, step-by-step. Make sure you can answer these questions (you'll need to know this for the exams):

- What is the name of the frame containing parameter `r` and variable `area`?
- When does the frame appear in the memory diagram?
- When do `r` and `area` appear in the frame?
- When does the frame disappear?
- What happens to `r` and `area` when the frame disappears?
- What is the name of the frame containing variable `result`?
- When does `result` appear in the frame?

Exercise 2 - A More Concise Version of `area_of_disk`

Step 1: In a `return` statement, the `return` keyword is followed by an expression. In the `area_of_disk` function, we can change the `return` statement, replacing variable `area` with the expression that calculates the area. So,

```
def area_of_disk(r):
    area = 3.14 * r ** 2
    return area

result = area_of_disk(5)
```

can be changed to:

```
def area_of_disk(r):
    area = 3.14 * r ** 2
    return 3.14 * r ** 2

result = area_of_disk(5)
```

Click the Edit code link in OPT, and make this change (highlighted in **boldface**) to your function definition. Click Visualize Execution and execute your modified script step-by-step.

Does `area_of_disk` return the same calculated area as the original version?

Step 2: Notice that in the modified function, variable `area` is never used after a value is bound to it. As such, `area` is is no longer needed, so we can simplify the function by deleting the statement: `area = 3.14 * r ** 2`:

```
def area_of_disk(r):
    return 3.14 * r ** 2

result = area_of_disk(5)
```

Make this change to your function definition. Execute the modified script, step-by-step, and observe the memory diagram. How do the frames in the memory diagram differ from those you observed in Exercise 1, Step 7?

Exercise 3 - Functions Can Call Functions

Step 1: A ring is a disk with a hole in the center. To calculate the area of a ring, we simply subtract the area of the hole from the area of the disk.

Edit the script in OPT so that it looks like this:

```
def area_of_ring(outer, inner):
    area_outer = 3.14 * outer ** 2
    area_inner = 3.14 * inner ** 2
    area = area_outer - area_inner
    return area

def area_of_disk(r):
    return 3.14 * r ** 2

result = area_of_ring(10, 5)
```

`area_of_ring` is a function that calculates and returns the area of a ring. This function has two parameters. Parameter `outer` is the radius of the ring and parameter `inner` is the radius of the hole.

Also, notice that the call to `area_of_disk` has been replaced by a call to `area_of_ring`. Don't delete the definition of `area_of_disk` - you'll need it in the next step!

Click Visualize Execution, and observe what happens when the script is executed step-by-step.

Step 2: The first two statements in `area_of_ring` calculate the areas of two disks. Why not call `area_of_disk` to do those calculations? Here is a revised function definition that does this. Notice how the two parameters of `area_of_ring` (`inner` and `outer`) are used as the arguments of the two calls to `area_of_disk`.

```
def area_of_ring(outer, inner):
    area_outer = area_of_disk(outer)
    area_inner = area_of_disk(inner)
    area = area_outer - area_inner
    return area

def area_of_disk(r):
     return 3.14 * r ** 2

result = area_of_ring(10, 5)
```

Use the OPT editor to make these changes (highlighted in **boldface**) to the function definition. Click Visualize Execution, and observe what happens when the script is executed step-by-step. Make sure you can answer these questions:

- What is the name of the frame containing parameters `inner` and `outer` and variables `area_inner`, `area_outer` and `area`?
- When does the frame appear in the memory diagram?
- When do `inner`, `outer`, `area_inner`, `area_outer` and `area` appear in the frame?
- When does the frame disappear?
- What happens to `inner`, `outer`, `area_inner`, `area_outer` and `area` when the frame disappears?
- What is the name of the frame containing parameter `r`?
- When does the frame appear in the memory diagram?
- When does `r` appear in the frame?
- When does the frame disappear?
- What happens to `r` when the frame disappears?
- What is the name of the frame containing variable `result`?
- When does `result` appear in the frame?

Step 3: Now that we understand how `area_of_ring` works, we can make it more concise (just like we did in Exercise 2), eliminating all the local variables. Edit the script once more, so that it looks like this:

```
def area_of_ring(outer, inner):
    return area_of_disk(outer) - area_of_disk(inner)

def area_of_disk(r):
     return 3.14 * r ** 2

result = area_of_ring(10, 5)
```

Execute the modified script, step-by-step, and observe the memory diagram. How do the frames in the memory diagram differ from those you observed in Step 2?

Exercise 4 - Defining Functions in a Module

While the Online Python Tutor is a great tool for visualizing the execution of short pieces of code, it is not a complete program development environment. On the other hand, any functions we type interactively in Wing IDE's Python shell (as you did in Exercise 1) are lost when we exit Wing IDE or restart the Python shell.

If we're writing more than a few lines of code, we typically prepare it using the IDE's editor and save it in a *source code file*.

A file containing a collection of Python function definitions is known as a *module*. You're now going to develop a module named lab2.py.

Step 1: Click the New button in the menu bar. A new editor window, labelled untitled-1.py, will appear.

Step 2: From the menu bar, select File > Save As... A "Save Document As" dialogue box will appear. Navigate to the folder where you want to store the module, type lab2.py in the File name box, then click the Save button.

Step 3: Here are the definitions of `area_of_disk` and `area_of_ring`. Type this code in the editor window, exactly as it's shown here. Once again, each statement in the function body should be indented four spaces.

```
def area_of_disk(r):
    """
    Return the area of a ring with radius r.
    """
    return 3.14 * r ** 2

def area_of_ring(outer, inner):
    """
    Return the area of a ring with radius outer.
    The radius of the hole is inner.
    """
    return area_of_disk(outer) - area_of_disk(inner)
```

We've made one addition to these functions. The triple-quoted string immediately after the function header is a *documentation string (docstring)*. This string summarizes what the function does.

Step 4: Click the Save button to save the edited module.

Step 5: Click the Run button. This will load lab2.py into the Python interpreter and check it for syntax errors. If any errors are reported, edit the function to correct them, then click Save and Run.

Step 6: You can now call the functions from the shell. Try these experiments:

```
>>> area_of_disk(5)
>>> area_of_ring(10, 5)
```

Exercise 5 - Importing Modules

Python is shipped with several modules (hence the phrase "batteries included" is often used when describing the language). The `math` module defines several common mathematical functions and two frequently-used values (`e` and `pi`). We can modify `area_of_disk` to use variable `pi` as a better approximation of the mathematical constant π (the changes are highlighted in **boldface**):

```
import math

def area_of_disk(r):
    """
    Return the area of a disk with radius r.
    """
    return math.pi * r ** 2
```

The statement:

```
import math
```

makes the definitions in `math` available throughout the module where `area_of_disk` is defined. To use variable `pi` in the calculation of the area, we must specify both the module name and the variable name; i.e., `math.pi`.

Make these changes to lab2.py, then use the shell to call `area_of_disk` and `area_of_ring`.

Have the values returned by these functions improved, now that you've replaced the literal value `3.14` with `math.pi`?

Exercise 6

The lateral surface area of a right-circular cone (a right cone with a base that is a circle) is given by the formula:

$$\pi r \times \sqrt{r^2 + h^2}$$

where r is the radius of the circular base and h is the height of the cone.

Define a function named `area_of_cone` in lab2.py that calculates and returns the lateral surface area of a right-circular cone. Here is the function header and docstring:

```
def area_of_cone(h, r):
    """
    Returns the lateral surface area of a right circular
    cone with height h and radius r.
    """
```

For the value of π, use variable `pi` from the `math` module. Python's `math` module also has a function that calculates square roots. To find out about this function, use Python's help facility. In the shell, type:

```
>>> help(math.sqrt)
```

Use the shell to interactively test your function.

Exercise 7

The volume of a sphere with radius *r* is given by the formula:

$$\frac{4}{3}\pi r^3$$

In lab2.py, define a function named `volume_of_sphere` that calculates and returns the volume of a sphere. The function has one parameter, the radius of the sphere.

Use the shell to interactively test your function.

Exercise 8

Assume that we have two spheres, one inside the other. In lab2.py, define a function named `hollow_sphere` that calculates and returns the volume of the larger sphere that is not occupied by the smaller sphere. This function has two parameters: the radius of the larger sphere and the radius of the smaller sphere. Your function must call the `volume_of_sphere` function you wrote for Exercise 7.

Use the shell to interactively test your function.

Extra Practice (do these on your own time, after you've completed the steps in the *Wrap Up* section on the next page)

- Without using Python or OPT, draw one or more memory diagrams that illustrate a typical call to your `area_of_cone` function. Use OPT to check if your diagrams are correct.
- Without using Python or OPT, draw memory diagrams that illustrate a typical call to your `volume_of_sphere` function. Use OPT to check if your diagrams are correct.
- Without using Python or OPT, draw memory diagrams that illustrate a typical call to your `hollow_sphere` function. Use OPT to check if your diagrams are correct.

Wrap-up

1. Remember to have a TA review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

2. Before you leave the lab, log in to cuLearn and submit lab2.py. To do this:

    - Click the Submit Lab 2 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag lab2.py to the File submissions box. Do not submit another type of file (e.g., a zip file, a RAR file, a .txt file, etc.)

    - After the icon for the file appears in the box, click the Save changes button. At this point, the submission status of your file is "Draft (not submitted)".

    - You can resubmit or delete the file by clicking the Edit my submission button. The page containing the File submissions box will appear.

        - To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

        - To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

    - Once you're sure that you don't want to make any changes, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading". **Make sure you do this before you leave the lab.**