**Carleton University**
**Department of Systems and Computer Engineering**
**ECOR 1041 — Computation and Programming — Winter 2021**

# Lab 3 - Functions

## Submitting Lab Work for Grading

You won't be submitting your lab work for Part 1 for grading. You'll only submit the code you write for Part 2.

Remember, you don't have to finish the lab by the end of your lab period. The deadline for submitting lab work to cuLearn for grading is 11:55 pm (Ottawa time) the day after your scheduled lab. Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the deadline.

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

## Part 1 - Understanding Function Definitions and Function Execution

### Objectives

To gain experience using Python Tutor to understand what happens when Python

- executes a function definition;
- executes a function call (a *call expression*).

### Exercise 1 - Defining a Simple Function and Tracing its Execution

**Step 1**: Recall that the area of a disk with radius $r$ is $\pi$ multiplied by the square of $r$, where $r$ is any positive real number. This can be represented by a mathematical function that maps each value of $r$ to the corresponding area (which is also a real number). A common notation for representing this function is:

$$f : r \rightarrow f(r), \text{where } f(r) = \pi r^2$$

The symbol $f$ represents the function and the symbol $f(r)$ is the value that $f$ associates with $r$; in other words, $f(r)$ is the value of $f$ at $r$.

To calculate the area of a disk that has a radius of 5.0, we substitute 5.0 for $r$ in the equation $f(r) = \pi r^2$ and evaluate the expression:

$$f(5.0) = \pi \times 5.0^2 = \pi \times 25.0 = 78.5398 \text{ (approximately)}$$

We can easily implement this function in Python[1]:

```
import math

def f(r):
    return math.pi * r ** 2
```

The function *header*, `def f(r):`, specifies that the function is named `f` and has one parameter, `r`, which is the radius of a disk. The function *body*, `return math.pi * r ** 2`, calculates and returns the disk's area.

Notice that the function uses variable `pi` from Python's imported `math` module. The value bound to this variable is an approximation of the mathematical constant π.

To calculate the area of a disk that has a radius of 5.0, we *call* the function with 5.0 as the *argument*:

```
f_at_r = f(5.0)
```

This code would be easier to understand if we used descriptive names for the identifiers; for example:

```
import math

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

area = area_of_disk(5.0)
```

You're now going to use Python Tutor to help you understand what happens as the Python interpreter executes each line of this source code, step-by-step.

Find the link Open Python Tutor in a new window in the *Lab Materials* section of the course cuLearn page. Launch Python Tutor, and verify that it is configured this way: "Write code in Python 3.6", "hide exited frames [default]", "render all objects on the heap (Python/Java)" and "draw pointers as arrows [default]". If necessary, select the correct options from the drop-down menus.

[1]The "official" Python style guide states that all `import` statements in a file (e.g., `import math`) should be at the top of the file, before any function definitions. (The style guide can be found here: https://www.python.org/dev/peps/pep-0008/.)

Type this *script* in PyTutor's editor window:

```
import math

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

area = area_of_disk(5.0)
```

Click the Visualize Execution button. Execute the code, one statement at a time, by clicking the Forward > button. Observe the memory diagram as the code is executed step-by-step. Answer each of these questions. (Although you won't be submitting your answers for grading, don't skip the questions just because your solutions won't be graded. The knowledge you gain is essential for the rest of the course.)

(a) What is the name of the frame containing variable `area_of_disk`?

(b) What object does variable `area_of_disk` refer to?

(c) What is the name of the frame containing parameter `radius`?

(d) The execution of what statement caused the frame to appear?

(e) Parameter `radius` refers to an object. What is the type and value of this object? The execution of what statement caused that object to be assigned to `radius`?

(f) How does PyTutor depict the value the function will return?

(g) The execution of what statement caused the frame containing `radius` to disappear?

(h) What is the name of the frame containing variable `area`?

(i) The execution of what statement caused `area` to appear in the frame?

(j) Variable `area` refers to an object. What is the type and value of this object?

**Exercise 2 - Learning More About Function Arguments**

Function arguments are expressions. When the function is called, these expressions are evaluated and the values are used as the arguments.

Click the Edit this code link in PyTutor and edit the script so that it looks like this (the changes are highlighted in **boldface**):

```
import math

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

area = area_of_disk(5.0)

area = area_of_disk(2.0 + 3)
diameter = 10.0
area = area_of_disk(diameter / 2)
area = area_of_disk(5)
```

Function `area_of_disk` is called four times. Notice that each call uses a different expression to calculate the function's argument; that is, `5.0`, `2.0 + 3`, `diameter / 2` and `5`.

Click Visualize Execution. Execute the code, one statement at a time, and observe the memory diagram. Answer these questions. (Again, you won't be submitting your answers for grading, but do the questions. You'll need to be able to answer similar questions throughout the course.)

(a) What value is assigned to parameter `radius` the first time `area_of_disk` is called?

(b) What value is assigned to parameter `radius` the second time `area_of_disk` is called?

(c) What value is assigned to parameter `radius` the third time `area_of_disk` is called?

(d) What value is assigned to parameter `radius` the fourth time `area_of_disk` is called?

**Exercise 3 - Using Functions as Building Blocks**

A ring is a disk with a hole in the center. `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. To calculate the area of the ring, the function first calculates the areas of the disk and the hole, then calculates the difference of the two areas:

```
def area_of_ring(outer, inner):
    return math.pi * outer ** 2 - math.pi * inner ** 2
```

We can simplify `area_of_ring` by using our `area_of_disk` function as a building block; that

is, `area_of_ring` will call `area_of_disk` to calculate both areas.

Here is the revised definition of `area_of_ring`. Notice how the two parameters of `area_of_ring` (`outer` and `inner`) are used as the arguments of the first and second calls to `area_of_disk`, respectively.

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

Edit the script so that it looks like this (the changes are highlighted in **boldface**):

```
import math

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

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

area = area_of_ring(10.0, 5.0)
```

Click Visualize Execution. Execute the code, one statement at a time, and observe the memory diagram. Answer these questions.

(a) What is the name of the frame containing parameters `inner` and `outer`?

(b) The execution of what statement caused the frame to appear?

(c) What is the name of the frame containing parameter `radius`?

(d) The execution of what expression caused the first appearance of this frame?

(e) The execution of what expression caused the second appearance of this frame?

(f) When Python evaluated the expression:

```
area_of_disk(outer) - area_of_disk(inner)
```

what was the value of the left-hand operand of `-` (the subtraction operator)? What was the value of the right-hand operand?

**Exercise 4 - The Point of No `return`**

Some programming languages don't have a `return` statement. Instead, the value returned by a function is the value of the last expression that is evaluated when the function executes. Is this true of Python?

Make these changes to the script:

- Delete the definition of `area_of_ring` and the statement that calls that function.
- Delete the `return` keyword in the body of `area_of_disk`.
- Add the statement, `area = area_of_disk(5)`

The script should now look like this (you don't need to type the comments):

```
import math

def area_of_disk(radius):
    math.pi * radius ** 2  # No return keyword.

area = area_of_disk(5)   # Add this assignment statement.
```

Click Visualize Execution. Execute the code, one statement at a time, and observe the memory diagram. Answer these questions.

(a) Does `area_of_disk` return the value of the expression

```
math.pi * radius ** 2
```

even though it isn't preceded by the `return` keyword?

If not, what is the type and value of the object returned by the function?

**Exercise 5 - Assigning Values to Function Parameters**

Delete all the code in PyTutor's editor window and type this script:

```
def cube(x):
    x = x ** 3

a = 2
cube(a)
```

When `cube` is called, the argument is variable `a`. The assignment statement in the body of `cube` calculates the value $x^3$, then assigns that value to parameter `x`. Will that value be copied to the function's argument, variable `a`? In other words, when a function assigns a new value to a

parameter, is that value also assigned to the corresponding argument?[2]

Let's run an experiment to answer these questions..

Click Visualize Execution. Execute the code, one statement at a time, and observe the memory diagram. Answer these questions.

(a) When `cube` is called, but before Python executes `x = x ** 3`, what value is bound to parameter `x`? What value is bound to variable `a`?

(b) After Python executes `x = x ** 3`, but before the function finishes executing, what value is bound to parameter `x`? What value is bound to variable `a`?

(c) After `cube` has finished executing (and its activation frame has disappeared), what value is bound to variable `a`?

(d) Based on your answers to (a), (b) and (c), what do you conclude? If a Python function assigns a value to a function parameter, is that value assigned to the corresponding function argument?

**Part 2 - Using Wing 101 to Define and Test Functions**

**Objectives**

To gain experience using Wing 101 to edit function definitions and interactively test functions.

**Getting Started**

Python Tutor is a great tool for visualizing the execution of programs, but it is not a complete program development environment, and it doesn't provide a way for us to save our code in a file. If we're writing more than a few lines of code, we typically prepare it using an editor and save it in a *source code* file.

**The "Rules"**

For this lab,

- you do not have to write *documentation strings* for your functions (brief descriptions of what the functions do and examples of how to use the shell to test the functions).
- **the body of each function must have exactly one statement: `return` followed by an expression.**
- your functions **must not** use local variables.
- your functions can assume that all function call arguments will be valid; for example, in Exercise 1, notice that the functions assume that their arguments are positive numbers.

[2] If Python permits this, it means a function can "return" values through its parameters, instead of executing a `return` statement.

**Exercise 1**

**Step 1**: Launch Wing 101.

**Step 2:** Click Wing's Create a new file button (the leftmost button in the toolbar - it looks like a piece of paper) or select File > New from the menu bar. A new editor window, labelled untitled-1.py, will appear.

**Step 3:** 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 file, then type lab3ex1.py in the File name: box. Click Save. Wing will create a new file named lab3ex1.py.

**Step 4:** In the editor window, type this code (which you used in Part 1):

```
import math

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

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

Don't copy/paste code from PDF files into the Wing 101 editor. Doing this may transfer non-displayable characters that Python doesn't recognize, triggering errors when you save the file or run the code. Because these characters aren't shown on the screen, locating them and removing them from the source code can be tedious.

After you've edited the code, save it in lab3ex1.py. To do this, click the Save button in the toolbar (it looks like a 3.5-inch floppy disk) or select File > Save from the menu bar.

**Step 5:** Click the Run button (the green triangle in the toolbar). This will load lab3ex1.py into the Python interpreter and check it for syntax errors. If any syntax errors are reported, edit the function to correct the errors, save the edited file, then click Run.

**Step 6:** Interactively test the functions by typing these *call expressions* (*function calls*) in the shell. Python will display the value that each function call returns; that is, the value of the call expression.

```
>>> area_of_disk(5)

>>> area_of_disk(5.0)

>>> area_of_ring(10, 5)

>>> area_of_ring(10.0, 5.0)
```

**Step 7:** Select File > Close from the menu bar. The editor window will no longer be displayed. (You can click the Open a file button on the toolbar - it looks like an open file folder - or select File > Open... from the menu bar if you want to reopen the file.)

**Exercise 2**

In Lab 2, you wrote Python code that converts 32 miles per Imperial gallon to (approximately) 8.83 litres per 100 km. In this exercise, you'll reimplement this code as a function that will convert any fuel efficiency measured in mpg to the equivalent metric fuel consumption.

**Step 1:** Create a new editor window and save it. Use lab3ex2.py as the file name.

**Step 2:** Type these two assignment statements and the function header. (One Imperial gallon is equal to approximately 4.54609 litres and one mile is equal to approximately 1.60934 km. Recall that the names of constant values in Python are, by convention, usually written in uppercase, with underscores separating the words.)

```
LITRES_PER_GALLON = 4.54609
KMS_PER_MILE = 1.60934

def convert_to_litres_per_100_km(mpg):
```

Complete the function definition; that is, write the function's body.

**Step 3:** Save the code, then click Run. Correct any syntax errors.

**Step 4:** Test your function by typing call expressions in the shell. What value does the function return when the argument is 32? Is this value correct? Does your function return the correct value when the argument is a `float` (32.0 instead of 32)? Does your function produce correct results when it is called with different positive arguments? How could you verify that these results are correct?

**Step 5:** Close the editor window for lab3ex2.py.

**Exercise 3**

In Lab 2, you wrote Python code that calculates the amount of money you'll have when money is deposited in a bank account that earns interest. In this exercise, you'll reimplement this code as a function.

---

Suppose you have some money (the *principal*) that is deposited in a bank account for a number of years and earns a fixed annual *rate* of interest. The interest is compounded *n* times per year.

The formula for determining the amount of money you'll have is:

$$amount = principal\left(1 + \frac{rate}{n}\right)^{n \times time}$$

where:

- *amount* is the amount of money accumulated after *time* years, including interest.
- *principal* is the initial amount of money deposited in the account
- *rate* is the annual rate of interest, specified as a decimal; e.g, 5% is specified as 0.05
- *n* is the number of times the interest is compounded per year
- *time* is the number of years for which the principal is deposited.

For example, if $1,500.000 is deposited in an account paying an annual interest rate of 4.3%, compounded quarterly (4 times a year), the amount in the account after 6 years is:

$$amount = \$1,500\left(1 + \frac{0.043}{4}\right)^{4 \times 6} \approx \$1938.84$$

**Step 1:** Create a new editor window and save it. Use lab3ex3.py as the file name.

**Step 2:** Type this function header:

```
def accumulated_amount(principal, rate, n, time):
```

Complete the function definition.

**Step 3:** Save the code, then click Run. Correct any syntax errors.

**Step 4:** Test your function by typing call expressions in the shell. Verify that your function returns the correct value for the example shown above. You should also test other cases. For example, what value would you expect the function to return if the principal is $0? If the interest rate is 0%? Does your function return correct values for these test cases?

**Step 5:** Close the editor window for lab3ex3.py.

**Exercise 4**

The lateral surface area of a right-circular cone (a right cone with a base that is a circle) can be calculated by the function:

$$f\ :\ h,\ r\ \rightarrow f(h,\ r);\ f(h,\ r)\ = \pi r \times \sqrt{r^2 + h^2}$$

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

**Step 1:** Use a calculator to determine the area of a right circular cone for various positive values of *r* and *h*. These will be your test cases for Step 5.

**Step 2:** Create a new editor window and save it. Use lab3ex4.py as the file name.

**Step 3:** Type this function header:

```
def area_of_cone(height, radius):
```

This function will return the lateral surface area of a right circular cone with the specified positive height and radius. Complete the function definition. 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:

```
>>> import math
```

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

**Step 4:** Save the code, then click Run. Correct any syntax errors.

**Step 5:** Use the shell to test your function. Use the test cases you prepared in Step 1. For each test, is the actual result (the value returned by the function) equal to the expected result (the value you calculated in Step 1)?

**Step 6:** Close the editor window for lab3ex4.py.

**Wrap Up**

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline. The submission deadlines for this lab are:

| Lab Section | Lab Date/Time | Submission Deadline (Ottawa Time) |
|---|---|---|
| 3A | Monday, 8:35 - 11:25 | Tuesday, Jan. 26, 2021, 23:55 |
| 2A | Tuesday, 14:35 - 17:25 | Wednesday, Jan. 27, 2021, 23:55 |
| 1A | Friday, 12:35 - 14:25 | Saturday, Jan. 30, 2021, 23:55 |
| 4A | Friday, 18:05 - 20:55 | Saturday, Jan. 30, 2021, 23:55 |

1. Login to cuLearn. Go to the cuLearn page **for your lab section** (not the main course page).
2. Click the link Submit Lab 3 for grading.
3. Click Add submission.
4. Submit your solutions for Part 2, Exercises 2, 3 and 4 by dragging the files lab3ex2.py, lab3ex3.py and lab3ex4.py to the File submissions box. Do not submit your file from Exercise 1 (lab3ex1.py).
5. Note that you have to confirm that your solutions are your own work before you can submit them. Click Save changes. The Submissions status page will be displayed. The line labelled *Submission status* will confirm that your submission is now *Submitted for grading*. The line labelled *File submissions* will list the names of the files you've submitted.
6. You are permitted to make changes to your solutions and resubmit the files as many times as you want, up to the deadline. Only the most recent submission is saved by cuLearn. To submit revised solutions, follow steps 7 through 9.
7. In cuLearn, click the link Submit Lab 3 for grading.
8. Click Remove submission to delete your previous submission. Click Continue to confirm that you want to do this.
9. Click Edit submission. Repeat steps 4 and 5 to submit your revised solutions.

**Extra Practice**

Use PyTutor to visualize the execution of your solutions to Exercises 2, 3 and 4. Remember, PyTutor doesn't have a shell. After copying your function definitions into the PyTutor editor, type assignment statements that call the functions and assign the returned values to variables.

History: Jan. 22, 2021 (initial release), Jan. 26, 2021 (Corrected typo. in submission instructions)