

Carleton University
Department of Systems and Computer Engineering
ECOR 1051 - Fundamentals of Engineering I

Lab 3 - Defining and Testing Functions

Objectives

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

Learning outcomes: 4, 5; Graduate attributes: 1.3, 5.3 (see the course outline)

Getting Started

Python Tutor, which you used in the last lab, is a great tool for visualizing the execution of programs, but it is not a complete program development environment and it does not provide a way for us to save our code in a file. If we are writing more than a few lines of code, we typically prepare it using an editor and save it in a *source code file* (with a “.py” file extension)

Launch Wing 101.

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).

Exercise 1

Step 1: 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 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 file, then type **lab3ex1** in the **File name:** box. Click the **Save** button. Wing will create a new file named **lab3ex1.py**. (Note that the “.py” file extension is added automatically to your filename by Wing101).

Do not type: lab3ex1.py

Doing so will create a file called lab3ex1.py.py

The TAs will not mark files that have the wrong file extension.

Step 3: In the editor window, type this code (which you used in Lab 2):

```
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)
```

After you've edited the code, save it. To do this, click the **Save** button in the toolbar or select **File > Save** from the menu bar.

Step 4: 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 5: In the previous step, when you ran your program, nothing would have happened (if you didn't make any typing mistakes). That is because you don't have any call expressions!

Test your function by writing the *main script* – below the function definition – made up of a series of call expressions that call your function with different values.

- Each call expression must save the returned value in the same variable

```
area = area_of_disk(5)
area = area_of_disk(5.0)
area = area_of_ring(10, 5)
area = area_of_ring(10.0, 5.0)
```

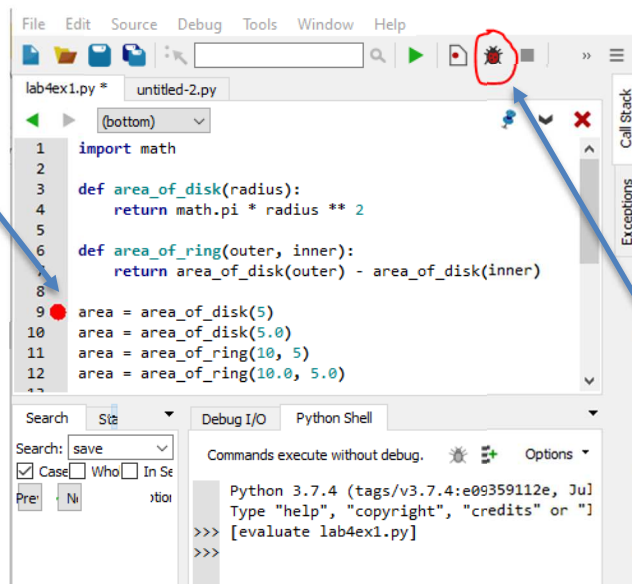
- For now, no `print()` statements are permitted!

Step 6: 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**.

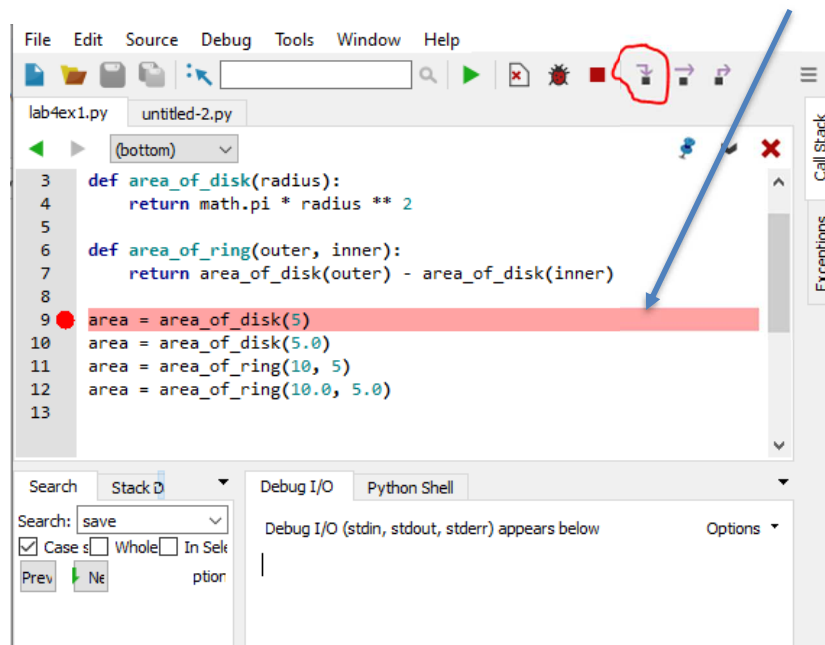
Step 7: In the previous step, when you ran your program, once again your program produces no meaningful output! Since you're not allowed to use `print()` statements yet, how are you going to see whether your program is working?

We're going to teach you how to use the debugger in the next series of steps. The debugger works like the **NEXT** button in Python Tutor allowing you to execute statements one step at a time.

To begin, position your mouse in the grey area on the left side of the screen, to the right of the line number 9. Click.



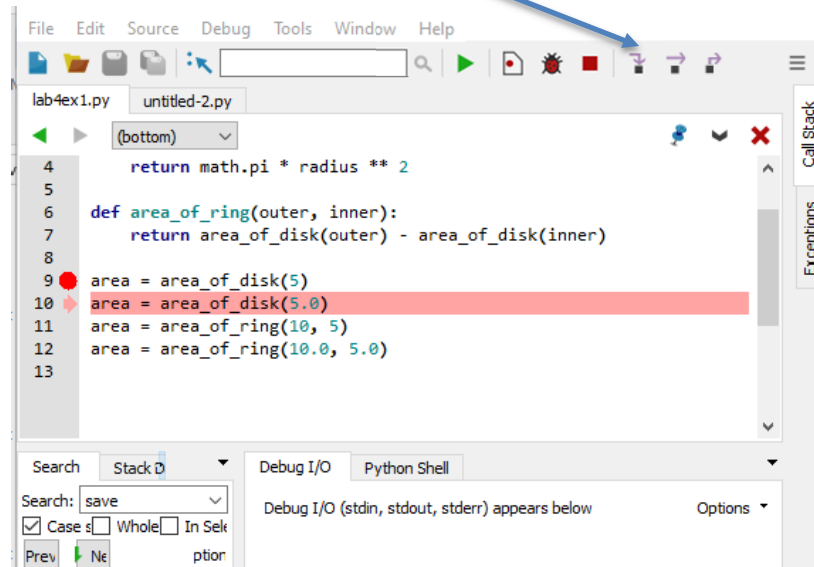
A red circle will appear, denoting that a **BREAKPOINT** has been set. When you run your program, the execution will now **BREAK** (i.e. stop) **before** line number 9 has executed; however, you must run your program “under the debugger” by **clicking on the ladybug** (not the green triangle). A red line will be shown where execution has stopped (below).



Remember that execution has stopped **BEFORE** the current instruction (in this case, Line 9, before the area variable has been assigned).

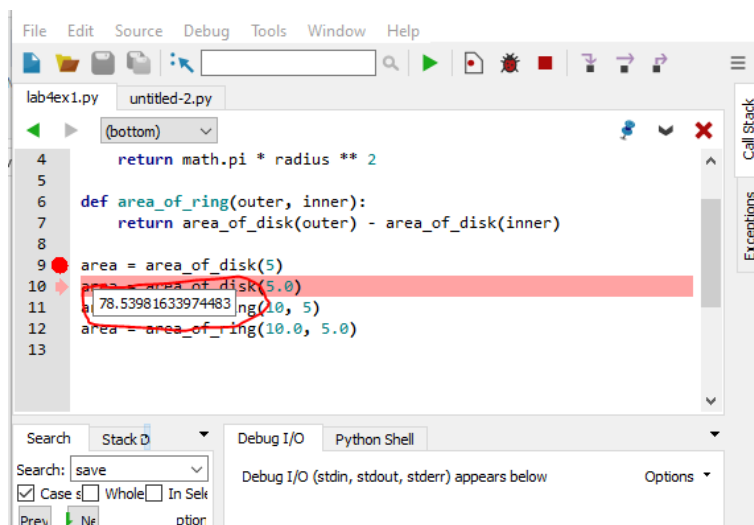
Step 8: Let's single-step (just like in Python Tutor). Let's execute Line 9, but stop before Line 10. To do this, click on the purple-bent-arrow icon (circled in red, in Step 7's diagram).

Repeatedly (but slowly) click on the purple single-step button until you reach Line 10 (as shown below).



Step 9: At this point, the variable `area` is bound to the area of a disk of radius 5.

At each breakpoint, the debugger allows us to “inspect” the current value of our variables. **Hover** your mouse over the variable `area` on Line 9 (or any line where it occurs). The variable's current value will be displayed (See below)



Step 10: Keep *single-stepping* until you have executed all four call expressions, inspecting the value of the variable `area` at each step to confirm that all four return values are correct (i.e. that the function always returns a correct value).

Step 11: Remove the breakpoint – using the same procedure you used in Step 7 to add it. Position your mouse in the grey area on the left side of the screen, to the right of the line number 9 and Click. The red dot will disappear.

Step 12: Add a `print()` statement after each call expression, to print the current value of `area`. Run your program (full-tilt, without breakpoints) by clicking on the green arrow. Notice how the printed values are the same as those you saw when you were debugging.

Step 13: Save your file.

Exercise 2

In a previous lab, you wrote Python code that converted miles-per-Imperial gallon to litres-per-100 km. In this exercise, you'll reimplement this code as a function.

Step 1: Create a new editor window (File->New) and save it as a file named `lab3ex2.py`.

REMEMBER: Only type `lab3ex2`. Do not type `lab3ex2.py`

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 entirely in uppercase.)

```
LITRES_PER_GALLON = 4.54609
KMS_PER_MILE = 1.60934

def convert_to_litres_per_100_km(mpg):
```

Write the body of the function to complete the function definition.

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

Step 4: Use the same procedure as in Exercise 1 to test your function (See Exercise 1, Step 5 to 13). Your call expressions should test:

- The value returned by the function when the argument is 32
- The value returned by the function when the argument is 0
- Whether the function works for integer arguments and real number arguments?

Exercise 3

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 will have is:

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

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:

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

Step 1: Create a new editor window and save it as a file named `lab3ex3.py`.

REMEMBER: Only type `lab3ex3`. Do not type `lab3ex3.py`

Step 2: Type this function header:

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

Write the body of the function to complete this function definition.

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

Step 4: Use the same procedure as in Exercise 1 to test your function (See Exercise 1, Step 5 to 13). Your call expressions should test your function

- For small simple values (use values like 10 and 2, so you can easily calculate the answer yourself)
- For “boundary values” (those likely to cause errors)
 - What is the principal is \$0?
 - What if the interest rate is 0%?

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: Create a new editor window and save it as a file named `lab3ex4.py`.

REMEMBER: Only type `lab3ex4`. Do not type `lab3ex4.py`

Step 2: 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 non-negative 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 3: Save the code, then click Run. Correct any syntax errors.

Step 4: Use the same procedure as in Exercise 1 to test your function. What values will you use to test your function?

Wrap Up

Create a new file called `lab3.py`. Combine your work as per the following diagram:

```
# Any and all import statements
# Function definition for Exercise 1
# Function definition for Exercise 2
...
# Main Script
# Call(s) of Function in Exercise 1
# Call(s) of Function in Exercise 2
...
```

Ensure that your code meets the posted marking rubrics for the labs.

Submit (only) file lab3.py.

You are required to keep a backup copy of (all) your work for the duration of the term.

Extra Practice

During the midterm and final exams, you will be expected to draw diagrams similar to those created by Python Tutor. Use Python Tutor to visualize the execution of your solutions to Exercises 2, 3 and 4.

Last edited: April 23, 2020