Carleton University
Department of Systems and Computer Engineering
ECOR 1041 - Computation and Programming

Lab 4 - Defining and Testing Functions

Objectives

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

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 lab4 in the File name: box. Click the Save button. Usually, Wing will create a new file named lab4.**py**.

Note that the "`.py`" file extension is added automatically to your filename by Wing101 in most cases. For some Macs, the "`.py`" extension is **not** automatically added, and you will have to type it. After saving, check your filename. Ensure that it is lab4.py and not lab4 or lab4.py.py. If the file does not have the "`.py`" extension, the debugger will not work. If the filename is not lab4.py, you will be ducted marks or get 0 as per the marking rubric!

Step 3: In the editor window, type this code. Make sure to include # (notation for Python's comments) in from of your name and student number. Otherwise, your code will not run, and you will get a zero for the lab.

```
# Name and student number
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 have edited the code, save it. To do this, click the Save button in the toolbar or select File > Save from the menu bar.

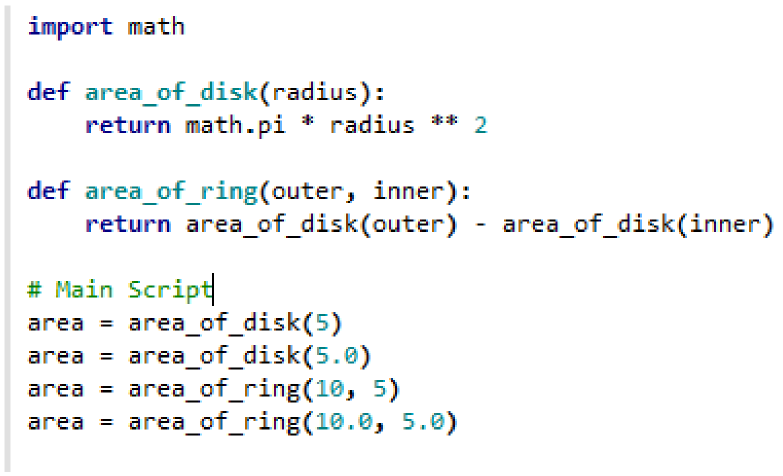
Step 4: Click the Run button (the green triangle in the toolbar). This will load lab4.py into the Python interpreter and check it for syntax errors. If any syntax errors are reported, edit the function to correct the errors, save the edited file, then click Run.

Step 5: In the previous step, when you ran your program, nothing would have happened (if you did not make any typing mistakes). That is because you do not 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)
```



```
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)

# Main Script
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)
```

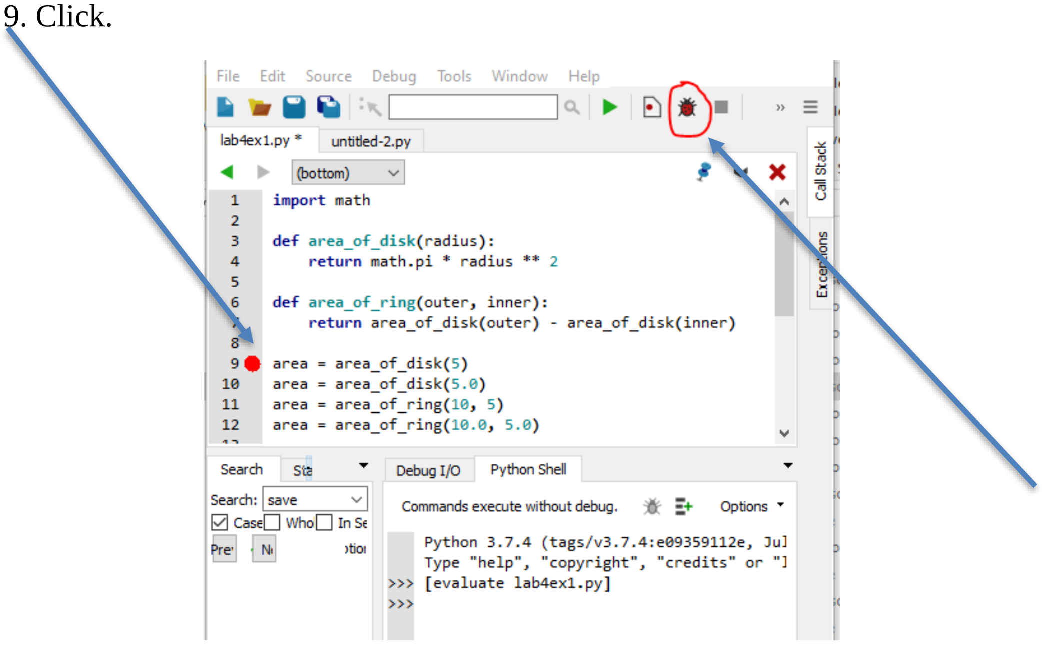
- For now, no `print()` statements are permitted!

Step 6: Click the Run button (the green triangle in the toolbar). This will load lab4.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 are not allowed to use `print()` statements yet, how are you going to see whether your program is working?

We are 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.

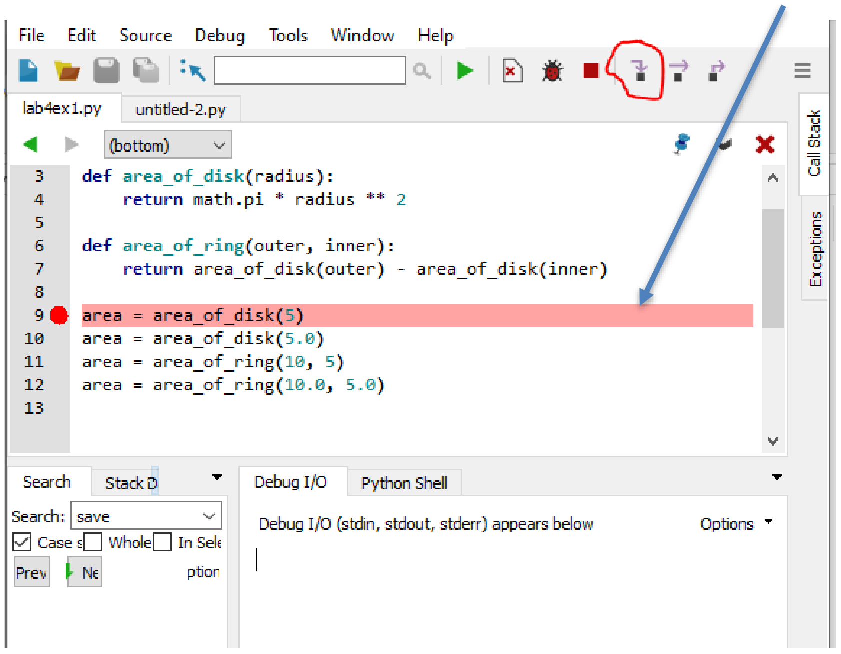
To begin, position your mouse in the grey area on the left side of the screen, to the right of the line number 9. Click.



```
1   import math
2
3   def area_of_disk(radius):
4       return math.pi * radius ** 2
5
6   def area_of_ring(outer, inner):
7       return area_of_disk(outer) - area_of_disk(inner)
8
9 ● area = area_of_disk(5)
10  area = area_of_disk(5.0)
11  area = area_of_ring(10, 5)
12  area = area_of_ring(10.0, 5.0)
```

```
    Python 3.7.4 (tags/v3.7.4:e09359112e, Jul
    Type "help", "copyright", "credits" or "]
>>> [evaluate lab4ex1.py]
>>>
```

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).

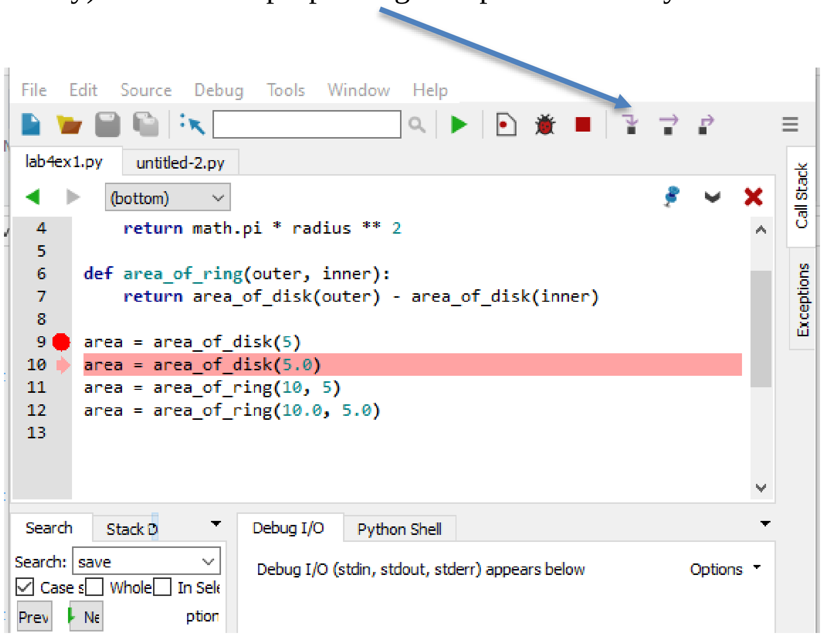


```
3   def area_of_disk(radius):
4       return math.pi * radius ** 2
5
6   def area_of_ring(outer, inner):
7       return area_of_disk(outer) - area_of_disk(inner)
8
9 ● area = area_of_disk(5)
10  area = area_of_disk(5.0)
11  area = area_of_ring(10, 5)
12  area = area_of_ring(10.0, 5.0)
13
```

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).

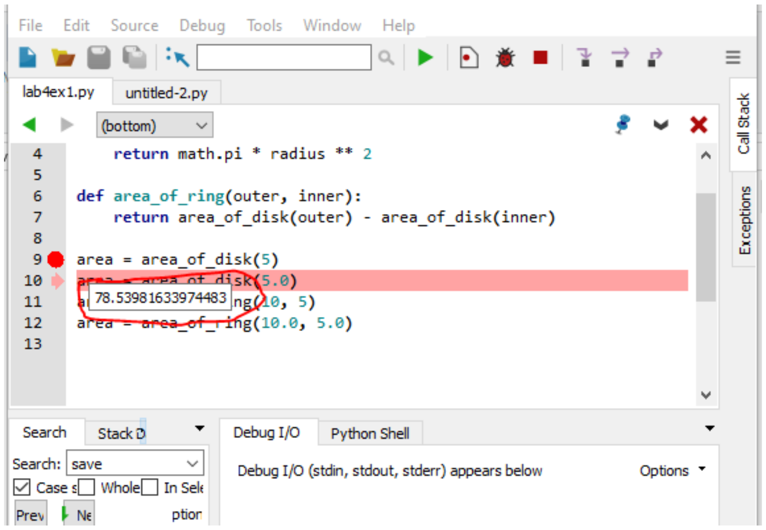


```
4        return math.pi * radius ** 2
5
6    def area_of_ring(outer, inner):
7        return area_of_disk(outer) - area_of_disk(inner)
8
9    area = area_of_disk(5)
10   area = area_of_disk(5.0)
11   area = area_of_ring(10, 5)
12   area = area_of_ring(10.0, 5.0)
13
```

Debug I/O (stdin, stdout, stderr) appears 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)



```
4        return math.pi * radius ** 2
5
6    def area_of_ring(outer, inner):
7        return area_of_disk(outer) - area_of_disk(inner)
8
9    area = area_of_disk(5)
10   area = area_of_disk(5.0)
11   area = area_of_ring(10, 5)
12   area = area_of_ring(10.0, 5.0)
13
```

Debug I/O (stdin, stdout, stderr) appears 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 (…/30)

In this exercise, you will reimplement Python code that converts miles-per-Imperial gallon to litres-per-100 km as a function.

Step 1: We will continue adding code to this same file, **lab4.py**

All your solutions to all the exercises in this lab **(and all future labs)** will be stored in one file, one after the other. By the end of the lab, your file should be organized as shown below. Broadly stating: (1) name and student number, (2) import statements, (3) function definitions and (4) the main script (made up of call expressions that test the use of those functions). Use comments to organize your file clearly.

#Name and student number

#Any and all import statements

# Function definition for Exercise 1

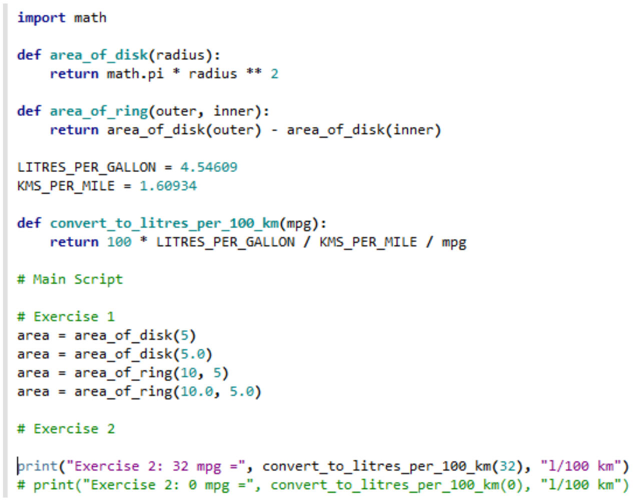
# Function definition for Exercise 2

…

Main Script

# Call(s) of Function in Exercise 1

# Call(s) of Function in Exercise 2



```
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)

LITRES_PER_GALLON = 4.54609
KMS_PER_MILE = 1.60934

def convert_to_litres_per_100_km(mpg):
    return 100 * LITRES_PER_GALLON / KMS_PER_MILE / mpg

# Main Script

# Exercise 1
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)

# Exercise 2

print("Exercise 2: 32 mpg =", convert_to_litres_per_100_km(32), "l/100 km")
# print("Exercise 2: 0 mpg =", convert_to_litres_per_100_km(0), "l/100 km")
```

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 35
- The value returned by the function when the argument is 0
- Whether the function works for integer arguments and real number arguments?

Make sure that your Python scrip includes the call expressions to test the returned values for the above-mentioned test cases.

Hint: Remember that your script running successfully to completion is required for your lab to get a mark other than 0! What should you do to ensure this happens for one of these test cases?

Exercise 3 (…/30)

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:

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

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:

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

Step 1: We will continue adding code to this same file, **lab4.py**

Step 2: Under the previous function definitions (but above the main script), 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 be at the bottom of the file, and should test your function

- Use small simple values, so you can easily calculate the answer yourself
- For "boundary values" (those likely to cause errors)
    - What if the principal is $0?
    - What if the interest rate is 0%?

Make sure that your Python script includes the call expressions to test the returned values for the above-mentioned test cases.

Hint: Remember that your script running successfully to completion is required for your lab to get a mark other than 0! What should you do to ensure this happens for one of these test cases?

Exercise 4 (…/40)

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

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

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

Step 1: We will continue adding code to this same file, **lab4.py**

Step 2: Under the other function definitions (but above the main script), type this function header:

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

This function will return the total area of a circular cone (base + lateral area) with the specified non-negative height and radius. **Write the body of the function** to 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? You need to use at least 3 different values

Write a comment in your Python script explaining which values you are using for testing. Remember that we use # to comment a line in Python.

Make sure you include the call expressions to test the returned values for the test cases you mentioned.

Wrap Up

- Make sure that you included your name and student number
- Check proper use constants (UPPER_CASE) and variables (lower_case) (There is a 10/100 deduction for misuse of UPPPER & lower case)
- Check the indents of the function bodies. (There is a 10/100 deduction for misuse of indentation)
- Check file organization: (1) imports, (2) **all** function definitions; (2) Main Script (There is a 10/100 deduction for not organizing the file according to the instructions)
- Confirm that your filename **lab4.py** matches exactly.
- Confirm that your .py script runs properly, otherwise the TA will also assign a zero.
- Submit the file on Brightspace.

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

Last edited: Dec 23, 2021