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

Lab 4 - Applying the Function Design Recipe

Objectives

To gain experience using *Practical Programming*'s function design recipe (FDR).

Submitting Lab Work for Grading

Before attempting Exercises 5, 6 and 7, read *Practical Programming*, Chapter 4, sections *Creating Strings of Characters* and *Operations on Strings*, pp. 65-68. This material will be introduced in Tuesday's (Feb. 2) lecture, so the deadline for students in lab section 3A (Monday) to submit their lab work to cuLearn has been extended to Wednesday night. For students in the other lab sections, the submission deadline is 11:55 pm (Ottawa time) the day after their 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: Functions that Work With Numbers (estimated time to complete: 2 hours or less)

Exercise 1

Step 1: Launch Wing 101. Create a new editor window and save it. Use `lab4ex1.py` as the file name.

Step 2: Here is the definition of function `area_of_disk`, which returns the area of a circle with a non-negative radius. (It should look familiar - you used it in Lab 3.)

```
import math

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

Type the code in the editor window. (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.)

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

Step 3: Add type annotations to the function header. Add a documentation string to the function definition. The docstring must include a sentence or two that describes what the function does (but not how it does it). Mention the function parameter in your description and describe the return value. (For examples of what we expect a docstring to contain, review last week's lecture materials and *Practical Programming*, section *Designing New Functions: A Recipe*, pp. 47-49.)

Below the description, type two or three example function calls and the return values we'd expect to see in the Python shell.

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

Step 4: Use the Python shell to test your function. After you've finished testing, close the editor window for lab4ex1.py.

Exercise 2

In this exercise, you'll use the function design recipe to design, code and test the definition of a function named `distance`. This function returns the distance between two points, given by the coordinates (x_1, y_1) and (x_2, y_2). The function parameters are the x and y values. (For those students who know Python, don't use lists or tuples to represent the points.)

If you don't remember the formula to use, check this page:

http://mathworld.wolfram.com/PythagoreanTheorem.html

Step 1: Create a new editor window and save it. Use lab4ex2.py as the file name.

Step 2: Type this function header and empty docstring.

```
def distance(x1, y1, x2, y2):
    """
    """
```

Step 3 (Examples): In the docstring, type two or three example function calls and return values. Use a calculator to determine the value the function should return for a given set of arguments.

Step 4 (Header): Add type annotations to the function header.

Step 5 (Description): In the docstring, write a sentence or two that describes what the function does (but not how it does it). Mention the parameters in your description and describe the return value.

Step 6 (Body): Design and code the function body. It must have **exactly one** statement: **`return`**

followed by an expression. Do not use local variables.

Hint: you'll need to use the `sqrt` function in the `math` module. See your solution to Lab 3, Part 2, Exercise 4 (the `area_of_cone` function).

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

Step 7 (Test): Use the Python shell to test your function (use the docstring examples). After you've finished testing, close the editor window for lab4ex2.py.

Exercise 3

In this exercise, you'll use the function design recipe to design, code and test a function named `area_of_circle`. This function takes two points: the center of a circle, (x_c, y_c), and a point on the perimeter, (x_p, y_p), and returns the area of the circle. The function parameters are the *x* and *y* values.

Step 1: Create a new editor window and save it. Use lab4ex3.py as the file name.

Step 2: Type this function header and empty docstring:

```
def area_of_circle(xc, yc, xp, yp):
    """
    """
```

Step 3 (Examples): In the docstring, type two or three example calls and return values. Use a calculator to determine the value the function should return for a given set of arguments.

Step 4 (Header): Add type annotations to the function header.

Step 5 (Description): In the docstring, write a sentence or two that describes what the function does. Mention the parameters and describe the return value.

Step 6 (Body): Design and code the function body. It must have **exactly one** statement: **`return` followed by an expression.** It must call the `area_of_disk` and `distance` functions you wrote for Exercises 1 and 2. Do not use local variables.

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

Step 7 (Test): Use the Python shell to test your function (use the docstring examples). After you've finished testing, close the editor window for lab4ex3.py.

Exercise 4

You've volunteered to repaint a room and will need to use a ladder when painting close to the ceiling. You know that a ladder has to be placed against a wall at an angle; otherwise, it will fall over. You're going to use the function design recipe to design, code and test the definition of a function named `height`. This function takes the length of a ladder, measured in metres, and the angle that it forms with the ground as it leans against the wall, measured in degrees. It returns the height reached by the ladder.

Step 1: Create a new editor window and save it. Use `lab4ex4.py` as the file name.

Step 2: Type this function header and empty docstring:

```
def height(length, angle):
    """
    """
```

Step 3 (Examples): In the docstring, type two or three example calls and return values. Use a calculator to determine the value the function should return for a given set of arguments.

Hint: the most challenging part of this exercise is the math, not the Python code. Use pre-university trigonometry to express the solution as a mathematical formula before you prepare the examples; for example, $f(l, a) = ...$, where l is the ladder's length and a is the angle, measured in degrees.

Step 4 (Header): Add type annotations to the function header.

Step 5 (Description): In the docstring, write a sentence or two that describes what the function does. Mention the parameters and describe the return value. Think carefully about the range of permitted values for the two parameters. Remember to document these in your docstring.

Step 6 (Body): Design and code the function body. It must have **exactly one** statement: **`return` followed by an expression.** Do not use local variables.

Hint: use the shell to investigate which trigonometric functions are provided by Python's `math` module:

```
>>> import math
>>> help(math)
```

Remember that functions in the `math` module are called this way:

```
math.function_name(argument_list)
```

not this way:

```
function_name(argument_list)
```

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

Step 7 (Test): Use the Python shell to test your function (use the docstring examples). After you've finished testing, close the editor window for lab4ex4.py.

Part 2: Functions that Process Text (estimated time to complete: 1 hour or less)

Exercise 5 (Adapted from *Practical Programming*, Chapter 4, Exercise 8)

Step 1: Create a new editor window and save it. Use lab4ex5.py as the file name.

Step 2: Type this code in the editor window:

```
def repeat(s: str, n: int) -> str:
    """ Return s repeated n times; if n is negative, return the
    empty string.

    >>> repeat('yes', 4)
    'yesyesyesyes'
    >>> repeat('no', 0)
    >>> repeat('no', -2)
    >>> repeat('yesnomaybe', 3)
    """
```

Step 3: Complete the examples in the docstring (type the strings that the second, third and fourth calls should return).

Step 4: Design and code the function body. It must have **exactly one** statement: **`return` followed by an expression.** Do not use local variables.

Step 5: Use the Python shell to test your function (use the docstring examples). After you've finished testing, close the editor window for lab4ex5.py.

Exercise 6 starts on the next page.

Exercise 6 (Adapted from *Practical Programming*, Chapter 4, Exercise 9)

Step 1: Create a new editor window and save it. Use lab4ex6.py as the file name.

Step 2: Type this code in the editor window:

```
def total_length(s1: str, s2: str) -> int:
    """ Return the sum of the lengths of s1 and s2.

    >>> total_length('yes', 'no')
    5
    >>> total_length('yes', '')
    >>> total_length('YES!!!!', 'Noooooo')
    """
```

Step 3: Complete the examples in the docstring (type the strings that the second and, third calls should return).

Step 4: Design and code the function body. It must have **exactly one** statement: **`return` followed by an expression.** Do not use local variables.

Step 5: Use the Python shell to test your function (use the docstring examples). After you've finished testing, close the editor window for lab4ex6.py.

Exercise 7

In this exercise, you'll use the function design recipe to design, code and test a function named `replicate`. This function has one argument, which is a string. It returns a new string that contains one or more copies of the argument. The number of copies is equal to the number of characters in the argument.

For example, when called this way: `replicate('a')`, the function returns `'a'`. The returned string contains one `'a'`, because the argument has one character.

When called this way, `replicate('abc')`, the function returns `'abcabcabc'`. This string contains three copies of `'abc'` because the argument has three characters.

Step 1: Create a new editor window and save it. Use lab4ex7.py as the file name.

Step 2: Type this function header and empty docstring:

```
def replicate(s):
    """
    """
```

Step 3 (Examples): In the docstring, type two or three example calls and return values.

Step 4 (Header): Add type annotations to the function header.

Step 5 (Description): In the docstring, write a sentence or two that describes what the function does. Mention the parameters and describe the return value.

Step 6 (Body): Design and code the function body. It must have **exactly one** statement: **`return` followed by an expression.** Do not use local variables.

Step 7 (Test): Use the Python shell to test your function (use the docstring examples). After you've finished testing, close the editor window for lab4ex7.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	Wednesday, Feb. 3, 2021, 23:55
2A	Tuesday, 14:35 - 17:25	Wednesday, Feb. 3, 2021, 23:55
1A	Friday, 12:35 - 14:25	Saturday, Feb. 6, 2021, 23:55
4A	Friday, 18:05 - 20:55	Saturday, Feb. 6, 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 4 for grading.
3. Click Add submission.
4. Submit your solutions by dragging the seven files containing your functions (lab4ex1.py, lab4ex2.py, lab4ex3.py, lab4ex4.py, lab4ex5.py, lab4ex6.py and lab4ex7.py) to the File submissions box.
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 4 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. 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. 30, 2021 (initial release)