# Lab Week 5 Grading Rubric and Instructions

This lab is assigned for Week 5 of COM S 127: Introduction to Programming.

This lab is due by the end of the lab period seven (7) days after the one it is assigned in. See the syllabus for details.

## Lab Objective

The purpose of this lab is for students to familiarize themselves with creating functions, and assigning the function output to a variable when the function returns.

## Instructions/ Deliverables

**NOTE**: These tasks can be completed in any order you like. See the **Grading Items** section below for the point distribution.

**CITATION**: Many of the exercises found here could possibly be seen as adaptations of exercises found in the online textbook "How to Think Like a Computer Scientist: Interactive Edition" By Jeffrey Elkner, Peter Wentworth, Allen B. Downey, Chris Meyers, and Dario Mitchell.

- Available: https://runestone.academy/ns/books/published/thinkcspy/index.html?mode=browsing
- Accessed: 2-2-2023
- The abbreviation 'thinkcspy' and the chapter/ section number will be used to indicate where similar exercises can be found. This citation will be placed next to the exercise title.
    - ex: [thinkcspy 2.13] indicates a similar exercise can be found in chapter 2, section 13.

**Reading:**

- Read Runestone chapters 7 and 8, and show the TA the notes you took in your Engineering Notebook for each chapter once you are done.
    - NOTE: You do not need to complete any of the exercises at the end of the chapter. However, it would be helpful to you in the long term if you were to do so.

**`Code Conversion:`**

- Review/ retrieve your code from lab week 3 - specifically `cubeVolume.py`, `cubeSurface.py`, `sphereVolume.py`, `sphereSurface.py`, `fahrenheitToKelvin.py`, `kelvinToFahrenheit.py`, `fahrenheitToCelsius.py`, `celsiusToFahrenheit.py`, `celsiusToKelvin.py`, and `kelvinToCelsius.py` - everything *except* `initials.py`.
- Convert each one of these scripts to a new script that uses a function to calculate its output, instead of performing a calculation 'in line' with everything else in your code.
    - Be sure to include your citations *inside* the function itself.
    - The function in each script should be named according to the file name. For example, the `cubeVolume.py` script should have a `cubeVolume()` function, which takes the previously input value as a parameter.
- Once you have converted all of your previous scripts to using functions, create three new modules:

    - `myShapes.py`
    - `myTemps.py`
    - `calculationTest.py`
- Copy and paste your shape-related functions into the `myShapes.py` module.
- Copy and paste your temperature-related functions into the `myTemps.py` module.
- Import both the `myShapes` module and the `myTemps` module into `calculationTest.py`.
- Create a `main()` function inside `calculationTest.py` as seen in the lecture slides, and in the example script below. This is where most of your code for this module should go.
- Inside the `main()` function of `calculationTest.py`, create a 'while loop' with a condition consisting of a boolean variable initially set to `True`.
    - Take input from the user to determine which of the functions in `myShapes` or `myTemps` they want to use, or if they want to quit.
        - Use conditional logic, depending on the previous input, to determine what to do next.
            - If the user wants to quit, set the variable used in the condition of the 'while loop' to `False` so that the loop terminates.
            - If the user wants to run one of the functions in `myShapes` or `myTemps`, take the appropriate input and pass that value to the appropriate function.
                - The function you call should then return its output back to `calculationTest.py` where you should print it out.
            - If the user does not input any valid choice, print out an 'error message' stating so.
    - **HINT**: This task will be very similar to the `luckyCalculator.py` assignment. The primary differences will be the use of the 'while loop' to keep the script running until the user decides to quit, and the use of modules to keep the various functions separate.
- Save your code to files called `myShapes.py`, `myTemps.py`, and `calculationTest.py`. Each of these files should include your name, code creation date, lab number, and *new* brief description in *each* file. These will be the files you show the TA to 'check off.'

**`drawMultipleTridecagons.py` [thinkcspy 6.13]**

- Make a copy of your `tridecagonTurtle.py` script from the previous lab, and save it as `drawMultipleTridecagons.py`.
- The purpose of this script is to create a new function which draws multiple tridecagons in a line - one after the other.
- Change your new `drawMultipleTridecagons.py` script so that it has a new function called `drawMultipleTridecagons()`. This new function takes five parameters, `s`, `x`, `y`, `nr`, and `sr`.
    - These parameters should be entered by the user as integer input.
    - The five parameters correspond to:
        - `s` - the length of the side of a regular tridecagon.
        - `x` - the 'x' coordinate of the first tridecagon to be drawn.
        - `y` - the 'y' coordinate of the first tridecagon to be drawn.
        - `nr` - the number of tridecagon repetitions to be drawn. For example, if `nr = 5`, then you draw five (5) tridecagons.
        - `sr` - the amount of space between tridecagon repetitions on the 'x' axis. For example if `x = 80`, `nr = 3`, and `sr = 50`, then there would be tridecagons starting at `x = 80`, `x = 130`, and `x = 180`.
    - Your new `drawMultipleTridecagons()` function should call your original `tridecagonTurtle()` function, passing in the appropriate parameters to the `tridecagonTurtle()` function. Your new `drawMultipleTridecagons()` function will draw the number of new tridecagons specified by the function parameters on the 'x' axis in

intervals also specified by the function parameters.

- Create a `main()` function inside `drawMultipleTridecagons.py` as seen in the lecture slides, and in the example script below.
- Be sure to move all of your screen object creation code inside this new `main()` function, as well as any other code that was not a part of your original `tridecagonTurtle()` function. Basically - *all* of your code should now be inside functions, and no longer at the global scope.
- Inside the `main()` function of `drawMultipleTridecagons.py`, make a call to your new `drawMultipleTridecagons()` function and observe the output.
- You should experiment with placing your turtle object creation code in two places - inside the original `tridecagonTurtle()` function, and inside the `main()` function. Be sure to remember how function scope works. If your turtle object creation code is inside the `main()` function, you will have to pass in a reference to your turtle object to the `drawMultipleTridecagons()` function, which will pass it to your original `tridecagonTurtle()` function.
    - **NOTE**: Either version of the above is fine for your 'check off' - you just cannot have your turtle object/ screen object creation code at the global scope.
- Save your code, including your name, code creation date, lab number, and brief description of what your code does, to a file called `drawMultipleTridecagons.py`.

# Files Provided

**None**

# Example Script

### myTest.py

```
# Matthew Holman                2-10-2023
# Lab Week 5 - A module to test another module's functions

import myFuncs

def getTwoInputs():
    a = int(input("Enter an integer: "))
    b = int(input("Enter an integer: "))
    return a, b

def main():
    running = True
    while running:
        choice = input("Choice?: [a]dd, [q]uit: ")
        if choice == "a":
            a, b = getTwoInputs()
            answer = myFuncs.add(a, b)
            print("The answer is: {0}".format(answer))
        elif choice == "q":
            print("Goodbye!")
            running = False
                else:
            print("ERROR: Please enter 'a' or 'q'")

if __name__ == "__main__":
    main()
```

**myFuncs.py**

```
# Matthew Holman                2-10-2023
# Lab Week 5 - A module to demonstrate functions

def add(a, b):
        return a + b
```

# Example Output

**Running: `python .\myTest.py`**

```
Choice?: [a]dd, [q]uit: a
Enter an integer: 2
Enter an integer: 3
The answer is: 5
Choice?: [a]dd, [q]uit: asdf
ERROR: Please enter 'a' or 'q'
Choice? [a]dd, [q]uit: q
Goodbye!
```

# Grading Items

- (**Attendance**) Did the student attend the lab meeting, or make arrangements to attend virtually via WebEx?: _____ / 1
- (**Reading**) Has the student read chapters 7 and 8 of the Runestone textbook and shown their notes in their Engineering Notebook to the TA?: _____ / 2
- (**Code Conversion**) Has the student completed the task above, and saved their work to files called `myShapes.py`, `myTemps.py`, and `calculationTest.py`?: _____ / 4
- (**drawMultipleTridecagons.py**) Has the student completed the task above, and saved their work to a file called `drawMultipleTridecagons.py`?: _____ / 3

**TOTAL _____ / 10**