Lab #4 Grading Rubric and Instructions

This is the fourth lab for the 'Introduction to Computer Programming' course.

Please see the syllabus for information about when the work in this lab is due.

Lab Objective

The purpose of this lab is for you to familiarize yourself with calling functions from modules and assigning the function output to a variable when the function returns. This lab will also let you practice using `if` statements.

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.

CITATION: Some of the exercises found here are completely original to the instructor.

- The abbreviation 'MH' will be used to indicate these exercises. This citation will be placed next to the exercise title.
    - ex: [MH]

Lecture Notes/ Supplemental Readings:

- 'Check off' your notes in your Engineering Notebook for the following material with the TA/ Instructor. These notes should already be done before the start of the lab period.
    - Runestone chapter 7
        - **NOTE**: You do not need to complete any of the exercises at the end of the Runestone chapters. However, it would be helpful to you in the long term if you were to do so.
        - **NOTE**: You only need to take notes on the Runestone readings if you did *not* take lecture notes in class. The contents of these chapters are what was covered in the lectures.
    - `Boolean_Equivalence_Patterns.pptx`
        - **Available**: `'Files' -> 'Supplimental_Slides_and_Files'` folder on Canvas
    - Introduction to Truth Tables - by: David Lippman
        - **Available**: https://math.libretexts.org/Courses/Prince_Georges_Community_College/MAT_1130_Mathematical_Ideas_Mirtova_Jones_(PGCC%3A_Fall_2022)/02%3A_Logic/2.02%3A_Introduction_to_Truth_Tables

`Code Conversion pt. 2`: [MH]

- If you have not converted the specified code from Lab Week 3 into functions in Lab Week 4 ("Code Conversion pt. 1"), you will need to do so now.
- Once you have converted all of your previous scripts to using functions, create five new modules:
    - `myShapes.py`
    - `myPhysics.py`
    - `myOhmsLaw.py`
    - `myFinances.py`
    - `calculationTest.py`
- Copy and paste your four shape-related functions into the `myShapes.py` module.
    - `areaOfRectangle.py`
    - `rectanglePerimeter.py`
    - `areaOfCircle.py`
    - `circleCircumference.py`
    - **NOTE**: Only copy the functions that calculate things. Do not include any `main()` functions or `if __name__ == "__main__":` code. You are *not* copying and pasting everything you did in the previous lab into just one file.
- Copy and paste your two physics-related functions into the `myPhysics.py` module.
    - `distanceSpeedTime.py`
    - `velocityAccelerationTime.py`
    - **NOTE**: Only copy the functions that calculate things. Do not include any `main()` functions or `if __name__ == "__main__":`

code. You are *not* copying and pasting everything you did in the previous lab into just one file.
- Copy and paste your three Ohm's Law-related functions into the `myOhmsLaw.py` module.
  - `calculateVoltage.py`
  - `calculateResistance.py`
  - `calculateCurrent.py`
  - **NOTE**: Only copy the functions that calculate things. Do not include any `main()` functions or `if __name__ == "__main__":` code. You are *not* copying and pasting everything you did in the previous lab into just one file.
- Copy and paste your two other useful functions into the `myFinances.py` module.
  - `annualPercentageRate.py`
  - `compoundAmount.py`
  - **NOTE**: Only copy the functions that calculate things. Do not include any `main()` functions or `if __name__ == "__main__":` code. You are *not* copying and pasting everything you did in the previous lab into just one file.
- Import the `myShapes.py`, `myPhysics.py`, `myOhmsLaw.py`, and the `myFinances.py` modules into `calculationTest.py` at the top of the file (but below the initial comment block with your name, date, etc.).
- Create a `main()` function inside `calculationTest.py` as seen in the lecture slides, activity 6.8.2 of the Runestone book, and in the `myTest.py`/ `myFuncs.py` example scripts below.
  - This new `calculationTest.py` script is where most of your new code for this exercise should go.
- Inside the `main()` function of `calculationTest.py`, create a '`while` loop' (like the ones seen in section 8.3 of the Runestone book) with a loop condition consisting of a new boolean variable, created before the loop, and initially set to `True`.
  - **HINT**: You may need to read section 8.3 of the Runestone book to familiarize yourself with how a `while` loop works if the code in the 'Example Script' section, below, is unclear to you.
- Inside your `while` loop, take input from the user to determine which of the functions in `myShapes.py`, `myPhysics.py`, `myOhmsLaw.py`, or `myFinances.py` 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.py`, `myPhysics.py`, `myOhmsLaw.py`, or `myFinances.py`, take the appropriate input and pass that input to the appropriate function.
    - The function you call should then return its output back to `calculationTest.py` where you should print it out.
    - You can take this input however you like, but you should probably create a re-usable function if you are doing the same input pattern over and over again.
  - 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.
  - While we may not have talked about 'while loops' in class by the time your lab section meets, the code in the 'Example Script' section below should 'plant the seed' of how they work.
- **HINT**: Please see the example code for `myTest.py` and `myFuncs.py` (in the 'Example Script' section) for details about what the code you will be creating for this exercise should look like.
- Save your code to files called `myShapes.py`, `myPhysics.py`, `myOhmsLaw.py`, `myFinances.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/ Instructor 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`.
- Give your new `drawMultipleTridecagons.py` script a `main()` function as you have done previously.
  - This is where you will take your input from the user, and then call the new function you will create for this exercise.
- The purpose of this script is to create a new function which draws multiple tridecagons in a line - one after the other. It will do this by calling your original tridecagon-drawing function multiple times in a loop.
  - It will be a function that calls another function!
- Update your new `drawMultipleTridecagons.py` script so that it has a new function called `drawMultipleTridecagons()`. This new function takes six parameters, `s`, `x`, `y`, `nr`, `sr`, and `t`.
  - The `s`, `x`, `y`, `nr`, and `sr` values should be entered by the user as integer input in the `main()` function.
  - The six parameters of your `drawMultipleTridecagons()` function 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`.
    - `t` - the turtle object you will create in `main()`, after creating your screen object.
- Your new `drawMultipleTridecagons()` function should call your original `tridecagonTurtle()` function (which should already be in your new file).
- You will take the appropriate input in your `main()` function, create the appropriate turtle/ screen objects, and then pass the appropriate arguments/ turtle to the `drawMultipleTridecagons()` function.
  - This function will then pass the necessary arguments/ turtle to your original `tridecagonTurtle()` function.
- After you have taken the appropriate input from the user, call new `drawMultipleTridecagons()` function which will then draw the number of new tridecagons specified by the function parameters on the 'x' axis in intervals also specified by the function parameters.
- **HINT**: You will need to place your `tridecagonTurtle()` function in a 'for loop,' inside `drawMultipleTridecagons()` and update the

arguments you pass to it with each loop iteration to make this work.
- **HINT**: You will be passing the turtle you create in `main()` to `drawMultipleTridecagons()`, which will then pass the same turtle into the original `tridecagonTurtle()` function.
- **HINT**: Be sure to move all of your screen object/ turtle creation code inside the new `main()` function, as well as any other code you use to take the user's input. *All* of your code should now be inside functions. Meaning that *none* of your code should be at the global scope any longer.
- 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`.

`findLeapYear.py`: [thinkcspy 7.10]

- Take input from the user in the form of one integer - `year`.
- Write a function that takes this integer as input.
- This function will be called `findLeapYear()`, and it will calculate whether the specified `year` is a leap year or not using the following algorithm from [thinkcspy 7.10]:
  - "The year is evenly divisible by 4;"
  - "If the year can be evenly divided by 100, it is NOT a leap year, unless;"
  - "The year is also evenly divisible by 400. Then it is a leap year."
  - If the `year` is a leap year, the function returns `True`, else it returns `False`.
- For this file, use a `main()` function, like the one seen in section 6.8 of the Runestone textbook.
  - Use the `if __name__ == "__main__":` version as seen in activity 6.8.2.
- Call `main()` inside the `if __name__ == "__main__":` statement.
- Collect your input inside the `main()` function.
- Call the new function you created for the file and pass your input value into the new function. Make sure to collect the return value from the function in a new variable back in `main()`.
  - Ex: `answer = findLeapYear(year)`
- Finally print out the results of the calculation.
  - When you print the answer, if `findLeapYear()` is `True`, you print `YES`, else print `NO`.
  - Do *not* print `True` or `False`.
- Save your code, including your name, code creation date, lab number, and brief description of what your code does, to a file called `findLeapYear.py`.

`adventureStory.py`: [MH]

- Use the Python programming language to write a short interactive 'story' using `if` statements to control the flow of the narrative, and the `input()` function to let the user decide what to do next.
- Below, is some example output of one such 'story' (you will need to use your imagination and write your own):

```
Welcome to the Adventure Story!
By: Matthew Holman

You find yourself in a spooky mansion with three hallways ahead of you.
Do you want to go [left], [right], or [forward]?: left
You walk down the left hallway and come across a locked door.
Do you want to [break] down the door, or [find] another way?: break
You break down the door and step inside - only to find that it's a spooky library!
Do you want to go [back], or [explore] the library?: explore
You have found a secret treasure map! Surely it will be worth something to someone!
Do you want to go [back], or [look] over the map?: back
You have returned to the entrance and left the spooky mansion.
Yay, you won the game!
```

- For this file, use a `main()` function, like the one seen in section 6.8 of the Runestone textbook.
  - Use the `if __name__ == "__main__":` version as seen in activity 6.8.2.
- Call `main()` inside the `if __name__ == "__main__":` statement.
- Write the code for your story inside the `main()` function.
  - You should use at least five (5) `if` statements to tell your story. Do not worry about using any other programming paradigms other than `if` and `input()`.
- **HINT**: If you are having trouble figuring out what to do, try to write the code that reproduces the 'story' above.
- Save your code, including your name, code creation date, lab number, and brief description of what your code does, to a file called `adventureStory.py`.

Attendance:

- If you have completed all of your tasks for the lab, you may work on any of the 'Additional Resources for Study' found in the Canvas announcement of the same name.
  - **NOTE**: If you leave early, you will not receive the 'attendance points' for the lab.

Optional Readings

NOTE: These readings are not required. However, they may provide a bit of interest/ insight into the broader world of Computer Science. Please complete the rest of your lab tasks before doing these readings. You do not need to take notes on these in your Engineering Notebook.

How to Spot Scams Online and IRL - by: Morgan & Morgan

- Available: https://www.forthepeople.com/blog/how-spot-scams-online-and-irl/

Scams: six of the most common tricks – and how to avoid them - by: Hilary Osborne

- Available: https://www.theguardian.com/money/2023/oct/07/scams-six-of-the-most-common-tricks-and-how-to-avoid-them

Tips on When to Start Applying for Jobs before Graduation - by: Resume Professional Writers

- Available: https://www.resumeprofessionalwriters.com/apply-for-jobs-before-graduation/

Python Modules and Packages – An Introduction - by: John Sturtz

- Available: https://realpython.com/python-modules-packages/

The Top 7... Worst jobs in the games industry - by: GamesRadarTylerWilde

- Available: https://www.gamesradar.com/the-top-7-worst-jobs-in-the-games-industry/

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(f"The answer is: {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
```

HINT: Notice how we can return multiple values from a function ala the `getTwoInputs()` function in `myTest.py`. What would you need to do if you only needed one input?

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

- (**Lecture Notes/ Supplemental Readings**) Has the student taken notes on the listed material, and shown their notes in their Engineering Notebook to the TA/ Instructor?: _____ / 20
- (**Code Conversion pt. 2**) Has the student completed the task above, and saved their work to files with the names specified in the exercise?: _____ / 30
- (**drawMultipleTridecagons.py**) Has the student completed the task above, and saved their work to a file called `drawMultipleTridecagons.py`?: _____ / 20
- (**findLeapYear.py**) Has the student completed the task above, and saved their work to a file called `findLeapYear.py`?: _____ / 10
- (**adventureStory.py**) Has the student completed the task above, and saved their work to a file called `adventureStory.py`?: _____ / 10
- (**Attendance**) Did the student attend the full lab meeting in person?: _____ / 10

TOTAL _____ / 100