Lab Week 11 Grading Rubric and Instructions

This lab is assigned for Week 11 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 to give students a review of functions and dictionaries, as well as practice with recursion.

Instructions/ Deliverables

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

Reading:

- Read the following articles, and prepare written summaries of each topic in your Engineering Notebook to show the TA.
    - Topic: Call Stack
        - https://en.wikipedia.org/wiki/Call_stack
        - https://eecs280staff.github.io/notes/02_ProceduralAbstraction_Testing.html
            - Just read the sections "Function Calls and the Call Stack," and "Function-Call Process."
            - CITATION: "These notes were written by Amir Kamil in Winter 2019 for EECS 280. They are based on the lecture slides by James Juett and Amir Kamil, which were themselves based on slides by Andrew DeOrio and many others."
            - https://creativecommons.org/licenses/by-sa/4.0/
                - No changes were made
    - Topic: Recursion
        - https://ptuszak.medium.com/a-beginners-guide-to-recursion-470b893024e8
- Read Runestone chapter 16, and show the TA the notes you took in your Engineering Notebook for this 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.

`dictAndFuncts.py`

- Carefully read the problem description in the `DictionariesAndFunctionsReview.pptx` file, and complete the task contained therein.
- **NOTE**: None of these functions are recursive in any way.
- Use a `main()` function in the way demonstrated in class.
    - Ex: `if __name__ == "__main__":` etc.
- See the **Example Output** section below for examples of what this script should do.
- Save your code, including your name, code creation date, lab number, and brief description of

what your code does, to a file called `dictAndFuncts.py`.

`countGoalRecursive.py`

- Consider the outline in the **Example Script** section below.
  - Complete the function called `countDownGoalRecursive()` such that it takes a value of `n`, which will be greater than the value of `goal`, and prints out a pattern like the one seen in the **Example Output** section below.
    - This function will involve a recursive call to `countDownGoalRecursive()`, and it will have a call to the `print()` function. These function calls will happen in a certain order. It is your job to determine this order, and figure out why this is.
- Use a `main()` function in the way demonstrated in class.
  - Ex: `if __name__ == "__main__":` etc.
- See the **Example Output** section below for an example of what the first function in this script should do.
- Once you have completed the `countDownGoalRecursive()` function above, consider the following: what would happen if you *reversed* the order of the calls to `countDownGoalRecursive()` and `print()`? For example, if you called the recursive step for `countDownGoalRecursive()` and then `print()`, you would get a certain output. However if you called `print()` and *then* recursive step for `countDownGoalRecursive()`, the output would change.
  - Make a second function, with a new, appropriate, and descriptive name, based on the change to the output observed in the point above. Call this function from `main()` and print out the new output after the output of the original `countDownGoalRecursive()` function.
- Save your code, including your name, code creation date, lab number, and brief description of what your code does, to a file called `countGoalRecursive.py`.
- **NOTE**: Once you have completed this task, your file will have **two** recursive functions inside of it, not just one as seen in the examples below.

Drawing

- Create a drawing, either by hand or with the a drawing program in the computer, where you trace the 'call stack' of the `catAttack.py` script below. This drawing should appear in your engineering notebook.
  - Each time a function is called, add a new step to the drawing with a single box - i.e. *push* the function onto the stack. This box should be labeled with the function name, and include the current values of its parameters. If there are no parameters, just include empty parentheses, as you would with a normal call to such a function. e.g. `main()`
  - Each time a function returns, add a new step to the drawing where the box that corresponded to the function that just returned is removed - i.e. *pop* the function off of the stack.
    - Each time the 'call stack' changes, you will re-draw the previous step in a new location, and either add or remove the appropriate box.
  - For the `print()` function (which would, indeed, be pushed to/ popped off the stack), just write `print()`, and what the `print()` function would print to the terminal inside the parentheses. e.g. `print("Meow T !")`
  - Do not include the `range()` and `len()` functions in your drawing.
    - Your drawing should only include boxes for `main()`, `catAttack()`, `catStrike()`, and `print()`.
  - **NOTE**: This drawing will depict a series of 'steps' of adding and removing function calls to and from the stack. It could end up being a drawing that is quite large, with many different

'steps' depicted. **This is NOT a trivial task**.

- **HINT**: You can use the VS Code debugger to trace the 'call stack' and the function variables.
- **HINT**: See the lecture slides about 'exceptions' for an example of what each step in this 'call stack' drawing might do.
- **HINT**: Have a *very* firm idea of how the stack works **before** you start your drawing. It will be very large (perhaps multiple pages), and you do not want to have to re-do it if you make a mistake.

`catAttack.py`

```
def catStrike(x):
    print("Meow", x, "!")

def catAttack(cat):
    for i in range(0, len(cat)):
        catStrike(cat[i])

def main():
    catAttack("TOM")

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

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.

5 ways to avoid being catfished - by: Malwarebytes Labs

- Available: https://www.malwarebytes.com/blog/news/2022/06/5-ways-to-avoid-being-catfished

The insider's guide to recursion interview questions - by: Educative

- Available: https://www.educative.io/blog/the-insiders-guide-to-recursion

When Should You Apply for a Summer Internship? (With 6 Tips) - by: Indeed Editorial Team

- Available: https://www.indeed.com/career-advice/finding-a-job/when-to-apply-for-summer-internship

Gaming Industry: The good the bad and the ugly - by: Priscilla Martin

- Available: https://towardsdatascience.com/gaming-industry-the-good-the-bad-and-the-ugly-47c8e1244e24

Files Provided

DictionariesAndFunctionsReview.pptx

Example Script

dictAndFuncts.py

```
def inputValidInteger():
    pass # Your code goes here

def calcAAA(a, b, c, d):
    pass # Your code goes here

# etc. (26 additional functions)
# Your code goes here

def main():
    a = inputValidInteger()
    b = inputValidInteger()
    c = inputValidInteger()
    d = inputValidInteger()

    calculations = {}
    calculations["AAA"] = calcAAA(a, b, c, d)
    # etc. (26 additional functions)
        # Your code goes here

    # Print the keys and values
        # Your code goes here

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

countGoalRecursive.py

```
def countDownGoalRecursive(n, goal):
    pass # Your code goes here!

def main():
    countDownGoalRecursive(5, 3)

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

Example Output

Running: **`python dictAndFuncts.py`**

```
input ‘a’: 1
input ‘b’: 1
input ‘c’: 1
input ‘d’: 1
AAA: 4
AAS: 2
```

etc.

Running: `python countGoalRecursive.py`

```
5
4
3
```

Grading Items

- (**Attendance**) Did the student attend the lab meeting, or make arrangements to attend virtually via WebEx?: ______ / 1
- (**Reading**) Has the student read the assigned articles, as well as chapter 16 of the Runestone textbook and shown their notes in their Engineering Notebook to the TA?: ______ / 1
- (**`dictAndFuncts.py`**) Has the student completed the task above, and saved their work to a file called `dictAndFuncts.py`?: ______ / 3
- (**`countGoalRecursive.py`**) Has the student completed the task above, and saved their work to a file called `countGoalRecursive.py`?: ______ / 3
- (**Drawing**) Has the student completed the drawing task above in their Engineering Notebook and shown it to the TA?: ______ / 2

TOTAL ______ / 10