Lab Week 7 Grading Rubric and Instructions

This lab is assigned for Week 7 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 practice with using/ creating/ modifying lists.

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: 2-26-2023
- 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 chapter 10, 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.

`findMinMax.py` [thinkcspy 10.31]

- Use a `main()` function in the way demonstrated in class.
    - Ex: `if __name__ == "__main__":` etc.
- Inside your `main()` function, call a new function which uses a 'while' loop to take in string input until the user enters `*`.
    - Each piece of input (except for the final `*`) should consist of string representations of integers.
    - When you are done, your function will return the list it takes in and you will assign this list to a new variable in main().
        - Thus, you will be left with a list of strings such as: `numbers = ["1", "2", "3", "4"]`
- Convert the strings in your list to integers, such that it will now be: `numbers = [1, 2, 3, 4]`
- Create a new function, called `findMin()`, which takes a list as its parameter.
    - Your new function will traverse the list it takes in from its parameter, and finds the

minimum value present in the list.
        - **NOTE**: You **cannot** use the `min()` function, the `.sort()` method, or any other built-in way to easily find a solution. You will have to use logic, not a trivial call to the Python API, to solve this problem.
    - Finally, return the value you find back to the `main()` function and print it out.
- Create a new function, called `findMax()`, which takes a list as its parameter.
    - Your new function will traverse the list it takes in from its parameter, and finds the *maximum value* present in the list.
        - **NOTE**: You **cannot** use the `max()` function, the `.sort()` method, or any other built-in way to easily find a solution. You will have to use logic, not a trivial call to the Python API, to solve this problem.
    - Finally, return the value you find back to the `main()` function and print it out.
- Save your code, including your name, code creation date, lab number, and brief description of what your code does, to a file called `findMinMax.py`.

`palindromeList.py` [thinkcspy 10.31]

- Use a `main()` function in the way demonstrated in class.
    - Ex: `if __name__ == "__main__":` etc.
- Inside your `main()` function, call a new function which uses a 'while' loop to take in string input until the user enters `*`.
    - Each piece of input (except for the final `*`) should consist of strings.
    - When you are done, your function will return the list it takes in and you will assign this list to a new variable in main().
        - Thus, you will be left with a list of strings such as: `palList = ["Cat", "Apple", "Orange", "Apple", "Cat"]`
- Create a new function, called `palindromeList()`, which takes a list as its parameter.
    - Your new function will traverse the list it takes in from its parameter, and determine whether its elements are 'palindromic' or not.
        - Ex: `palList = ["Cat", "Apple", "Orange", "Apple", "Cat"]` => `True`
        - Ex: `palList = ["Cat", "Orange", "Apple", "Apple", "Cat"]` => `False`
        - **NOTE**: You **cannot** reverse the list, create a new list and compare it to the old list, or use any built-in/ trivial methods to solve this problem. **However** You *may* use a single '`for`' loop to traverse the list.
        - **HINT**: How would you compare list indices at the start/ end of the list in a single loop iteration?
    - Finally, return the `True`/ `False` value you find back to the `main()` function and print it out.
- Save your code, including your name, code creation date, lab number, and brief description of what your code does, to a file called `palindromeList.py`.

`statisticsList.py` [thinkcspy 10.31]

- Use a `main()` function in the way demonstrated in class.
    - Ex: `if __name__ == "__main__":` etc.
- Inside your `main()` function, call a new function called `generateInput()`. The `generateInput()` function should be placed *outside* your main function - at the global scope.
    - This function should produce a list of random length between 200 - 500 (inclusive) elements.
    - The list created above should contain random integers between 1 - 2000 (inclusive).
    - Once the list has been created, return the list to the `main()` function and assign it to a

variable.

- Create two new functions: `findMean()` and `findMedian()`, and call them in your `main()` function.
    - Each function should take the random list created earlier as its input.
    - The `findMean()` function should find the mean of all the values in the list, and then return this value back to `main()` and assign it to a variable there.
        - You should research and cite what a statistical mean is inside your function.
        - Do not use the `sum()` function, or any trivial calls to the Python API to complete this task. Rather, iterate through the input list and sum the values that way.
    - The `findMedian()` function should find the median of all the values in the list, and then return this value back to `main()` and assign it to a variable there.
        - You should research and cite what a statistical median is inside your function.
        - You may use the `.sort()` list method to help you find the answer.
        - Do not use the bitwise negation (tilde ~) operator for this. Just find the middle value if the input list length is odd, or the two middle values if the input list length is even and average those two values.
- Once you have both values returned from their respective functions, print them out for the user:
    - Ex: `print("Mean: {0} Median: {1}".format(mean, median))`
- Save your code, including your name, code creation date, lab number, and brief description of what your code does, to a file called `statisticsList.py`.

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.

Machine learning, explained - by: Sara Brown - April 21, 2021

- Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

Evolutionary computation - by: engati.com

- Available: https://www.engati.com/glossary/evolutionary-computation

How To Get a Job in Computer Science (With Job Salaries) - by: Indeed Editorial Team - March 16, 2023

- Available: https://www.indeed.com/career-advice/finding-a-job/job-in-computer-science

Making Video Games Is Not a Dream Job - by: wired.com - May 14, 2021

- Available: https://www.wired.com/2021/05/geeks-guide-jason-schreier/

Files Provided

None

Example Script

exampleFunction.py

```
# Matthew Holman               2-26-2023
# Lab Week 7 - An example script layout

def listInput():
        inputList = []
        # NOTE: You will need to figure out how to take the user's input.
        #       This should continue until the user enters '*'
        return inputList

def exampleFunction(lst):
        answer = False
        # NOTE: You will need to manipulate the contents of lst in some way.
        #       For example, if this function checks if lst is 'palindromic,'
        #       and, indeed it is, then answer = True
        return answer

def main():
        inputList = listInput()
        answer = exampleFunction(inputList)
        print("The answer is:", answer)

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

Example Output

Running: `python .\exampleFunction.py`

```
Enter a string (* to stop): apple
Enter a string (* to stop): cat
Enter a string (* to stop): orange
Enter a string (* to stop): cat
Enter a string (* to stop): apple
Enter a string (* to stop): *
The answer is: True
```

NOTE: This example output conforms to the `palindromeList.py` exercise above, but the operation depicted here will be similar for the `findMinMax.py` exercise as well. Feel free to format things however you wish for either exercise, but the input/ output operations should make sense and look nice for the user.

Grading Items

- (**Attendance**) Did the student attend the lab meeting, or make arrangements to attend virtually via WebEx?: _____ / 1
- (**Reading**) Has the student read chapter 10 of the Runestone textbook and shown their notes in their Engineering Notebook to the TA?: _____ / 3
- (**`findMinMax.py`**) Has the student completed the task above, and saved their work to a file called `findMinMax.py`?: _____ / 2
- (**`palindromeList.py`**) Has the student completed the task above, and saved their work to a file called `palindromeList.py`?: _____ / 2
- (**`statisticsList.py`**) Has the student completed the task above, and saved their work to a file called `statisticsList.py`?: _____ / 2

TOTAL _____ / 10