CSCI 248 **Homework #9**
Spring 2022 **Due: 10 pm Sunday, April 3**

This assignment contains both written problems (W) and programming problems (P). For written problems, you will **type** your answers into the output file: **`hw9-output.txt`**. Remember to put a blank line between Problem 1 and Problem 2, etc., with all of the problems in order.

Reminder: Make sure you review the academic honesty section of the course syllabus before working on any assignment. For each problem in which you write a program, make sure you include complete header comments, as specified in Homework #1.

Reminder: Make sure you review the academic honesty section of the course syllabus before working on any assignment. For each problem in which you write or modify a program, make sure you include complete header comments, as specified in Homework #1.

Problem 1: (W) Write a code fragment that sets **integer** variables named **`hour`** and **`minute`** appropriately **based on a string** called **`time`**. For example, if **`time`** contains the value **`'11:55'`**, **`hour`** will set to 11 (**not `'11'`**) and **`minute`** will be set to 55 (**not `'55'`**). If **`time`** contains the value **`'5:00'`**, **`hour`** will set to 5 and **`minute`** will be set to 0. Note that you want **integer** answers, not strings. Hint: Use **`split`**.

Problem 2: (W) Suppose a list called **`name_list`** contains a list of names. For example, the list *might* be defined as follows:

```
name_list = ['Picasso', 'Monet', 'Da Vinci', 'Rembrandt']
```

Given that the list has **already** been created, write an **efficient** code fragment that uses a **single** loop to compute and prints the total length (in characters) of all the names in the list. For the sample list above, the output would be as follows:

```
There are 29 characters in the list.
```

Your code fragment should work even if the names in the list change. That is, you **cannot** assume that the list always has the 4 elements shown above. Reminder: Use a ***single*** loop, not nested loops.

IMPORTANT NOTE for the programming problems: Read **all** of the instructions for each problem before beginning. In some cases, I have given you a partial solution, and you are **required** to start with that partial solution.

Problem 3: (P) Writing a list function

Copy the file **`~lwilson/python/homework/count_evens.py`**, to your current directory.

Add to the file a function called **`count_evens`** that takes one parameter called **`num_list`** representing a list of integers. The function must return the ***count*** of the number of even elements in the list. For example, if the list contains [2, 3, 1, 8, 7, 4], the function will return 3 since there were 3 **even** numbers in the list. Give your function a proper docstring.
Hint: **`for num in num_list:`**

Run the program to test your function. Remember to add complete header comments before submitting your work.

Problem 4: (P) Computing the average

On multiple occasions, we have discussed how to compute the sum and/or average of a set of data. Your task here is to write a user-friendly program that computes the average of the numbers in a file, where there are (or could be) **multiple values per line**. For example, the first two lines could be as follows:

```
22 5.8 24.8
19
```

Specifically, you must write a program that takes one **command-line argument** that represents the name of the data file. The program then will compute and report the average of the file's elements as follows:

```
This program computes the average of the values in nums1.txt.

The average of the 8 values was 20.9.
```

You may assusme that the file contains at least one number, so there is no concern about dividing by 0. Your program must follow standard practices for command-line arguments, including an appropriate **usage message** in the format shown in the notes. Following standard practices, your program must handle file and data conversion exceptions appropriately.

Hints: Use `split`, and remember that `split` returns a list of **strings**. The list of strings returned by `split` will be your **only** use of lists in this problem.

Test your program using both `nums1.txt` and `nums2.txt`, which I have provided in the `~lwilson/python/homework` directory, where the second file should demonstrate handling of a data conversion error. Your sample output should demonstrate also that your program follows standard practices for command-line arguments. Remember to **show the command line** with the output.

Problem 5: (P) Computing the median

Copy the file `~lwilson/python/homework/test_median.py` to your current directory.

Write a function called `median` that takes a list of numbers as its one parameter and returns the median value. (As noted below, you will add your function to a program I have written.) If the number of items in the list is odd, the median is the middle value in a sorted list; if the number of items in the list is even, the median is the average of the two middle values in a sorted list. For example, the median of [4, 7, 1, 8] is 5.5 and the median of [1, 6, 2, 5, 12] is 5. The function should **not** modify the original list. Remember to include a suitable docstring, as you should with **all** functions other than `main`.

To help you understand this problem, here is rough pseudocode for this function:

```
make a sorted copy of the number list (and use that copy in remainder of the program)
if (the number of list elements is odd)
    median = the middle value in the list
else
    median = the average of the two middle values in the list
return median
```

Hints: Remember the **integer division** operator and the `len` function. You can read how to sort the list in the Chapter 7, Part 2 notes. Think carefully how to compute **where** the middle values are in an arbitrary list. You may find it helpful to draw some pictures of a list and its indexes. See two examples below.

0	1	2	3	4
8	12	13	20	24

0	1	2	3	4	5	6	7
2	7	10	14	17	19	23	30

Add your **`median`** function to **`test_median.py`** and run it to test your function. Look carefully at the output to make sure that your program does **not** modify the list! Remember to add complete header comments before you submit **`test_median.py`**.

Challenge Problem #7: (P) Finding prime factors

First, write a Python function that takes a positive integer *n* as a parameter and returns a list containing all the numbers in the prime factorization of *n*. (The prime factorization of a positive integer *n* is the unique list of prime numbers whose product is *n*.) For example, if *n* is 72, the resulting list is [2, 2, 2, 3, 3] since 2 × 2 × 2 × 3 × 3 = 72. If *n* is 5, the resulting list is [5].

Next, **use your function** as part of a complete, user-friendly program that prompts the user for a positive integer and then prints the prime factors for that integer. Your sample output should demonstrate thorough testing of your function.

Challenge Problem #8: (P) Working with a 2-D list (array)

The file **`~lwilson/python/homework/Womens-BBall-Points.csv`** contains data on the points scored by the TLU women's basketball team in recent years. We want a program that computes the points per game (ppg) for each year of data in the file.

I have developed a partial program in **`~lwilson/python/homework/read_BB_points.py`** that uses *problem decomposition* to divide the program into multiple functions. I have provided a function that reads the file and puts the data into a 2-D list called **`array`**. In each row, the first element in the list is a string containing the year represented by the rest of the row, while the rest of the row contains the TLU scores for the games played. (Remember that a ***row*** in a 2-D list is simply a 1-D list.) Here is an example of the first row of the array:

```
['2013-14', 85, 58, 37, 99, ...]
```

Your task is to write a function called **`compute_average`** that takes **`array`** as a parameter and returns a (different) **2-D list** containing the year and points-per-game average on each row. For example, the first row of the result will have the values ['2013-14', 72.2]. You will add your function to my partial solution to **complete** the program.

Hints: Remember to skip the first element in each row when computing the average, since the first element contains the year. Your function will start with an empty list **`[]`** and add rows to it (using **`append`**) as it determines them. Yes, you can append a 1-D list when working with a 2-D list.