15-110: Principles of Computing, Spring 2018

Programming Assignment 4

Due: Tuesday, February 13 by 9PM

Note: You are **responsible for protecting your solutions** to the following problems from being seen by other students both physically (e.g., by looking over your shoulder or verbal discussion) and electronically. In particular, since the lab machines use the Andrew File System (AFS) to share files worldwide, you need to be careful that you do not put files in a place that is publicly accessible.

If you are doing the assignment on the Gates-Hillman Cluster machines we use in the lab or on `unix.andrew.cmu.edu`, please remember to have your solutions inside a `private` folder (which is under your home directory). Our recommendation is that you create a `pa4` folder under `~/private/15110` for this assignment. That is, the new directory `pa4` is inside the directory named `15110`, which is inside the `private` directory.

Overview

For this assignment, you will create a Python file for each of the problems below. You should save all of these files in a folder named `pa4`. Once you have every file, you should zip up the `pa4` folder and submit the zipped file on Autolab.

This assignment will make you practice the iteration and make you learn how to write the corresponding Python code to a given algorithm. In addition to writing the code, you should test your code on multiple inputs, for which you independently know the correct output (e.g., by plugging the inputs into a calculator).

IMPORTANT: Set Up Instructions for PA4

Before you start these problems, make sure you have a copy of `sieve.py` from our course website in your `pa4` directory.

Problems

1. [3 points] Prime Factorization.

   Every integer greater than 1 can be expressed as the product of one or more prime numbers (which may be repeated). For example, 60 = 2 × 2 × 3 × 5, and the integers 2, 3, and 5 are all prime. This is called the number's prime factorization, and it is unique for each integer number of at least 2.

   An integer *n*'s prime factorization (*n* ≥ 2) can be calculated using the following algorithm:

   1. Set *dividend* equal to *n*.
   2. Let *primelist* be equal a list of all primes less than or equal to *n*. (Hint: call the `sieve` function and store what it returns in *primelist*.)
   3. Set *possible_factor* to the first element of *primelist*. Then remove it from *primelist*.
   4. Set *factors* to be an empty list.
   5. While *dividend* is not 1, do the following:
      a. If *possible_factor* is a divisor of *dividend*:
         i. Append *possible_factor* onto the list *factors*.
         ii. Set *dividend* equal to *dividend* divided by *possible_factor* (using integer division).
      b. Otherwise, (if you did not execute the substeps i and ii, above):
         i. Set *possible_factor* to the first element of *primelist*. Then remove it from *primelist*.
   6. Return the list *factors* as your result.

   Implement **this algorithm** as a Python function called `factor(n)` (stored in `factor.py`). Start your file with the line:

   ```
   from sieve import sieve
   ```

   and start your function on the next line. This import line will give you access to the `sieve` function in the `sieve.py` file. You can now call the `sieve` function as if it were in your `factor.py` file.

   Example Usage:

   ```
   -bash-4.2$ python3 –i factor.py
   >>> factor(60)
   [2, 2, 3, 5]
   >>> factor(45)
   [3, 3, 5]
   >>> factor(13)
   [13]
   >>> factor(15110)
   [2, 5, 1511]
   ```

2. [2 points] Harmonic Mean

The harmonic mean provides the truest average in certain situations. In physics, if a vehicle travels at 60 miles/hour for some distance d miles, then the same distance at 40 miles/hour, and then the same distance at 30 miles/hour, then its average speed is the harmonic mean which is 40 miles/hour which is the speed the vehicle could travel for 3d miles to have the same travel time.

The harmonic mean of a list of *n* positive numbers x_0, x_1, ..., x_{n-1} is equal to

$$\frac{n}{\frac{1}{x_0} + \frac{1}{x_1} + \cdots + \frac{1}{x_{n-1}}}$$

In the file `mean.py` define the function `mean(numlist)` that takes a non-empty list of **positive** numbers called *numlist* and returns the harmonic mean of the numbers in *numlist* accurate to 3 decimal places.

Use the following algorithm. Think about what division you should use in this algorithm.

1. Set *n* equal to the length of the number list.
2. Set *sum* equal to 0.
3. For each element *x* in *numlist*, do the following:
    a. Add 1/*x* to *sum* and store the result back into *sum*.
4. Set *result* equal to *n* divided by *sum*.
5. Return *result* rounded to the nearest 3 decimal places.

To round a number, use the `round` function which requires two arguments: the number you want rounded, and an integer indicating the maximum number of decimal places you want the number rounded to. For example:

```
round(123.456789, 4) is 123.4568
round(123.456789, 10) is 123.456789
round(123.9999, 2) is 124.0
```

Example Usage:

```
-bash-4.2$ python3 –i mean.py
>>> mean([60, 40, 30])
40.0
>>> mean([110])
110.0
>>> mean([50, 30])
37.5
>>> mean([10, 20, 40, 80, 160])
25.806
```

3. [2 points] Index of Minimum Element

   In `min_index.py`, define a Python function `min_index(datalist)` that returns the index of the minimum element in a non-empty *datalist* of elements. If the minimum element occurs more than once in *datalist*, your function should return the index of the first occurrence.

   Start with the algorithm presented in class and first convert it to find the minimum instead of the maximum. Then make an adjustment to it so that when you find a new minimum, you also keep track of its location or index. Then once the entire list is examined, you return the index of the minimum instead of the minimum itself. Be sure to test your function carefully on a wide array of lists of various sizes (except an empty list). Remember that lowercase strings are compared alphabetically.

   Example Usage:

```
-bash-4.2$ python3 –i min_index.py
>>> min_index([42, 15, 32, 78, 59, 29])
1
>>> min_index([30, 20, 10])
2
>>> min_index([99])
0
>>> min_index([10, 20, 30, 15, 25])
0
>>> min_index([60, 50, 40, 40, 40, 70])
2
>>> min_index(["homer", "marge", "bart", "lisa", "maggie"])
2
```

4. [3 points] Bubble Sort

   There are many algorithms for sorting the elements of a list. One of these is Bubble Sort. The way Bubble Sort works is that it scans through the list x_0, x_1, x_2, x_3, …, x_{n-1}, comparing each pair of adjacent values (x_0 with x_1, x_1 with x_2, x_2 with x_3, etc.). For each pair that is in the wrong order (i.e. $x_i > x_{i+1}$), these values are swapped. As a result, the first pass through the list causes the largest value to "bubble up" to the last position of the list. This process is repeated n-1 times which will cause the list to be sorted. (Do you see why?) This is not the most efficient implementation of Bubble Sort, but we'll start with this for now.

   (a) (1 point) In the file `swap.py`, write a function `swap(datalist, i, j)` that has three parameters: a data list and two indices of elements in the list, i and j. The function should swap the data in positions i and j in the data list. Swapping is like juggling. When you juggle, you toss one ball up into the air (freeing up one hand), you toss the other ball into your free hand, and then you catch the ball in the air with your other hand which is now free. To swap two values stored in variables, copy the first value into a temporary variable,

then copy the second value into the first variable, then copy the value in the temporary variable into the second variable.

Example Usage:

```
-bash-4.2$ python3 –i swap.py
>>> mylist = [10, 20, 30, 40, 50]
>>> swap(mylist, 1, 4)
>>> mylist
[10, 50, 30, 40, 20]
```

(b) (2 points) Now we will implement Bubble Sort which will use the swap function from part (a). In the file `sort.py` define a Python function `bubble_sort(datalist)` that **modifies** *datalist* by rearranging its elements so they are sorted in "ascending order" (from smallest to largest) using the Bubble Sort algorithm below. Do NOT use the built-in sort function. Start your file with the line:

```
from swap import swap
```

and then write the definition for your function afterwards.

1. Set *n* equal to the length of *datalist*.
2. Repeat the following *n*-1 times:
    a. For *index* in the range from 0 to *n*-2 inclusive, do the following:
        i. Compare *datalist*[*index*] with *datalist*[*index*+1]. If they are not in the correct order, then swap them. (Call the `swap` function you wrote above. What arguments do you use?)
3. Return *datalist*.

Be sure to test your function on more than just the one case shown below.

Example Usage:

```
-bash-4.2$ python3 –i sort.py
>>> mylist = [5, 2, 6, 3, 9, 1, 7, 8, 4]
>>> bubble_sort(mylist)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Submission

You should now have the `pa4` folder that contains the following six Python files:

`sieve.py, factor.py, mean.py, min_index.py, swap.py, sort.py`

Zip up the folder and submit the zipped file named as `pa4.zip` on Autolab.