University of Saskatchewan
Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 5

List Usage and Simple Dictionaries

Date Due: 5:00pm Friday Oct 25 **Total Marks: 19**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.5+
- You may not, in part or in whole, submit any code that was written by generative AI tools, such as ChatGPT
- **Assignments must be submitted to Canvas (the course website).**
- **Canvas will not let you submit work after the assignment hand-in closes.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (4 points):

Purpose: Practice using list comprehensions.

The following code extracts the city populations that are at least 10000 from the list of lists `populationsSK` and stores them in a new list `citiesSK`.

```
populationsSK = [ ['Pelican Narrows', 2703],
     ['Saskatoon', 222035],
     ['Moose Jaw', 33617],
     ['La Ronge', 5905] ]
citiesSK = []
for x in populationsSK:
     if x[1] >= 10000:
          citiesSK.append(x)
```

At the end of the program, `citiesSK` is the list: `[['Saskatoon', 222035], ['Moose Jaw', 33617]]`

(a) Modify the code so that it is an **equivalent** program in which the list `citiesSK` is created using a list comprehension instead of a loop.

(b) Using a similar list comprehension, write an expression that creates a list-of-lists containing the full information for all the towns whose population is STRICTLY smaller than 40000.

(c) Using a list comprehension, write an expression that creates a list of all the names of the cities and towns (the new list should contain ONLY the names, but no population data)

(d) Using a list comprehension, write an expression that creates a list-of-lists containing the full information only for the towns and cities whose names are more than 8 characters (including blanks).

What to Hand In

Hand in your solution in a file called `a5q1.py`.

Evaluation

- 4 marks, 1 for each comprehension

Question 2 (6 points):

Purpose: To practice your ability to modify lists and compute values with lists

For this question, we have a full year's worth of temperature data for the city of Saskatoon. But there's a problem: the temperatures are all in degrees Fahrenheit! In order to make this data meaningful to Canadians, we need to convert the data to degrees Celsius and then compute the average temperature for each month.

Starter File

You have been provided a starter file that contains the daily high temperatures for Saskatoon from the year 2023. The data is in a list of lists. Each sublist represents all the data for one month, with the first sublist being the data from January, the second being the data from February, and so on. The values in each sublist are temperatures measured in degrees Fahrenheit.

From here, you will write three separate functions to perform three tasks, as described below.

Converting units

Write a function that accepts a single temperature value in degrees Fahrenheit, and returns the matching value in degrees Celsius. This function should be very simple; it's just a **helper function** that you will use later.

Converting all the data

Write a function that accepts the list-of-lists of temperature data. For each temperature value in each sublist, the function should REPLACE that value with its matching value in degrees Celsius. Use your helper function to perform each calculation.

This function does not create any new lists and does not have a return value. None is needed. It simply modifies the existing lists of temperature values.

Calculating Monthly Averages

Write a function that accepts the list-of-lists of temperature data, and that returns a list containing the average temperature for each month in the data.

Recall that each sublist contains all the data for a single month, and since there are 12 such sublists, the list you return should have exactly 12 numbers in it.

Program Output

At the very bottom of your program, call the functions you made above, using their return values appropriately. Then use the list returned by the last function to print out the average temperature for each month in a readable format.

Sample Run

Here is an example of how your program's console output might look (there will be several more lines of ouput, we're just showing the first few months to save space).

```
Saskatoon average temperature highs, 2023
-------
Average temperature in January: -8.39
Average temperature in February: -8.39
Average temperature in March: -6.25
...
Average temperature in December: 1.13
```

Yes, in this particular dataset, it appears that the average temperatures for January and February are nearly identical. I was surprised too, but it seems to be a legitimate result. Verify it by hand if you're still unconvinced.

What to Hand In

Hand in your solution in a file called `a5q2.py`.

Evaluation

- 2 marks for overall correctness of output
- 4 marks for correctly following the design of the 3 functions

Question 3 (4 points):

Purpose: To practice reading from a file and using a dictionary as a mapping

Caching is a very powerful technique in computer science. The technique simply involves recording the result of a complex calculation rather than re-computing that result every time it is needed. It turns out that **dictionaries** are a great data type to use as the cache in this context.

It may or may not surprise you to hear that there is no known formula for determinining whether or not a given number N is **prime** or not. More or less, the only way to know is to try dividing N by all integers between 1 and itself and seeing if any of them yield a remainder.[1] For large N, this is a time-consuming computation, and therefore a good example of something that might be cached for some problems.

For this question, your task is to write a program that takes a given set of numbers, determines whether each of them is prime, and caches the result using a dictionary.

Provided Files

On the course website, you will find a starter file that already contains an implementation of a function called `is_prime()`, which accepts a positive integer and returns a Boolean value indicating whether or not that integer is prime. You can keep this provided function and use it in your program.

You will also find a sample input file, `numbers.txt`. It contains some positive integers, one per line of the file. You can use this to test your program, but keep in mind that your program must work for ANY similarly-formatted input file.

Write a Function

Write a function that takes a file name as its only parameter. The function should open the given file, read the numbers in it, and call the provided `is_prime()` function to determine whether each number is prime. The results of this process should be stored in a **dictionary**. The **keys** for this dictionary should be the numbers from the file, and the **value** for each key should be the Boolean result indicating whether the matching key is a prime number or not. The function should **return** this dictionary.

Print the results

Call the function you wrote, using `"numbers.txt"` as the argument. Print the returned dictionary to the console.

Sample Run

Your output might look something like this:

```
{1: False, 2: True, 3: True, 4: False, 5: True, 6: False, 7: True, 8: False, 9: False,
 10: False, 11: True, 15: False, 99: False, 100001: False,  100019: True,
 99999989: True, 99999991: False}
```

What to Hand In

- A file called `a5q3.py` containing your finished program, as described above.

[1] Some clever optimizations are possible to speed things up slightly, but none really change the fundamental approach.

Evaluation

- 4 marks for correctness of:
  - reading the file
  - calling the provided function on each number
  - caching the results in the dictionary

Question 4 (3 points):

All programs that you submit for this assignment will be assessed with regard to their **style and readability**. Make sure that all of your code:

- includes your name, nsid, and student number as comments
- uses informative variable names
- makes effective use of formatting and white space
- includes a `docstring` for all **function definitions** in the program
- for longer programs, uses helpful and concise single-line comments where appropriate

What to Hand In

There is nothing to hand in for this question. It's simply describing how the rest of your work will be evaluated with regard to style and readability. It is possible to earn these marks even if you don't hand in every other question in this assignment, but you must hand in enough work such that your style and readability can be adequately assessed.

Evaluation

- 3 marks for consistently following the principles listed above

Question 5 (2 points):

In your submission for this assignment, you have an opportunity to earn something we'll call "the wow factor". To earn the wow factor, you must do something creative that goes beyond the expectations laid out in the assignment. This could be an extra feature for one of the previous questions, or it could be an extra little program that you submit that builds on the skills used in the assignment.

You should not aim to do a lot of work for this, but should instead demonstrate in a simple and effective way the depth of your understanding. In short, take ownership of your work and "impress us".

Telling you exactly what to do to earn "the wow factor" would defeat the entire purpose, but here are some very general ideas.

- improve the aesthetic appeal of your program(s)
- improve the user-friendliness of your program(s)
- remove a simplifying assumption and handle the resulting added complexity
- compare two different ways of doing the same task, and submitting data/analysis on which was better
- just doing something generally cool

One quick warning: using fancy extra code libraries in order to AVOID a key learning objective from the assignment is not impressive, it's actually anti-impressive. Focus instead on solving problems using simple clean foundational programming skills. Using extra libraries is fine if they let you do something BEYOND what the assignment asked, but not if they're replacing a core assigned task.

What to Hand In

A plain text file called `wow.txt` describing what you did that you think merits the wow factor. One or two sentences is enough. If you create any additional code files, hand those in too.

Evaluation

- 2 marks for something impressive