CS 112, Lab 9 – Exercise (due Tues 7/6, 11:59pm)
Dictionaries, File I/O

Files:

- Create your own file with our convention (*userID_2xx_L9.py)*.
- You should also download tester file from blackboard for testing
- Run the tester as always: `python3 testerL9E.py gmason76_2B5_L9.py just these funcs`

As this is an **Exercise**, you can read any and all materials, ask us questions, talk with other students, and learn however you best learn in order to solve the task. Just create your own solution from those experiences, and turn in your work.

Reminder: you can run individual inputs, such as `counts("mississippi")`, by loading your file into interactive mode with the **-i** flag. You should be using this technique on projects as well.

```
demo$ python3 -i gmason_2B5_L9.py
>>> counts("mississippi")
{'m': 1, 'p': 2, 'i': 4, 's': 4}
```

Dictionaries serve a special niche of problem type. Being able to pair up "keys" with "values", and yet disallowing duplicate entries with matching keys, helps us organize data by the exact keys we want to use to initiate searching and inspecting our information.

We will solve some problems where perhaps there are solutions that could have been based upon lists, but hopefully the dictionaries approach will make the task simpler.

Turning It In

Add a comment at the top of the file that indicates your name, userID, G#, lab section, a description of your collaboration partners, as well as any other details you feel like sharing. Once you are done, run the testing script once more to make sure you didn't break things while adding these comments. If all is well, go ahead and turn in just your one .py file you've been working on over on BlackBoard to the correct lab assignment. We have our own copy of the testing file that we'll use, so please don't turn that in (or any other extra files), as it will just slow us down.

What can I use?

There are no restrictions on what functions to use on this lab – use this time to learn how to build, navigate, and modify dictionaries, and minimally how to read a file.

`counts(xs):` Consider a sequence of values, **`xs`**. It can contain duplicates, and we'd like to know how many of each present value there are. Construct and return a dictionary whose keys are the things found in the sequence, and whose corresponding values are the counts of occurrences.

- **`xs`** :: sequence of values. (It could be a list, a string, or other things…)
- Return value: a dictionary of things and their number of occurrences in **`xs`**.
- Examples:
  - `counts([1,1,1,2,3,3,3,3,5])` → `{1: 3, 2: 1, 3: 4, 5: 1}`
  - `counts("abracadabra")` → `{'r': 2, 'd': 1, 'c': 1, 'b': 2, 'a': 5}`

`weeklies(plants_d):` Consider a dictionary, **`plants_d`**, where the keys are names of plants, and the values are descriptions of how often to water them. Search through the entire structure for all plants that need to be watered **`"weekly"`**, put them into a list, **`sort()`** the list, and return it.

- **`plants_d`** :: dictionary of plant names to watering instructions.
- Return value: sorted **`list`** of plants that need watering **`"weekly"`**.
- `>>> weeklies({'shamrock':'weekly', 'cactus':'monthly', 'rose':'weekly', 'succulent':'biweekly'})`
  `['rose', 'shamrock']`
- `>>> weeklies({'fern':'weekly', 'shamrock':'weekly', 'carnation':'weekly'})`
  `['carnation', 'fern', 'shamrock']`

`closest(d, what, here):` Consider a dictionary that has pairs of integers as the keys, to represent a spot on a 2D grid (like pixels on a screen or integer points on the real numbers plane). The associated values will be strings, describing something that is present at that location. Using the distance formula between two points (see Wikipedia for a refresher if needed), complete the **`closest`** function that looks through the data for the spot closest to **`here`** that contains a **`what`**. For instance, "I'm at (2,3). Where's the closest gas station?" could be phrased as **`closest(d, "gas station", (2,3))`**. Don't worry about ties; anything equally closest will do (but we'll test with no ties to keep this short).

- Parameters:
  - **`d`** :: dictionary where keys are pairs of integers and values are strings.
  - **`what`** :: string describing a point of interest.
  - **`here`** :: tuple of coordinates of where we currently are.
- Returns: a pair of integers of the closest what to here. When no **`what`** is found, return the **`None`** value. (If we had learned about exceptions it would have been a great time for one here!)
- Examples:

```
>>> d = {(3,1):'gas', (1,4):'gas', (2,1):'food', (5,5):'food'}
>>> closest(d,"gas",(2,2))    #closest thing isn't gas.
(3, 1)
>>> closest(d,"gas",(5,5))
(1, 4)
>>> closest(d,"food",(5,5))
(5, 5)
>>> closest(d,"food",(1,4))
(2, 1)
>>> print(closest(d,"hotel",(1,4)))
None
```

`file_counts(filename):` Re-implementation of the counts function that obtains a list of numbers from a file. In the file, every single line will contain a single integer and nothing else. The last character will be a newline.

- **`filename`** :: a string indicating the name of a file in the current directory.
- Return value: a dictionary of int as keys and # of occurrences as the values.
- `file_counts("file1.txt")` → `{100:3, 3:1, 9:2}`
- `file_counts("file2.txt")` → `{1:1, 2:1, 3:1, 4:1, 5:1}`

file1.txt

```
100
100
3
100
9
9
```

file2.txt

```
1
2
3
4
5
```