Problem

Our task is to read in a CSV file from user input. The file contains pokemon data, with columns denoting different statistics, and rows denoting individual pokemons. Once read in, our task is to process this data in such a way that we create a 2D dictionary, indexed by type in the first dimension, and pokemon name in the second dimension. When we finish processing our data, we need to accept queries from user input, and then, based on that input, calculate the type that corresponds to the maximum average value for that stat, and print it to the screen.

Possible Solution 1: Usage of 2D Dictionaries as Required

Pre-Processing Stage

1. Read in a file name from the user

2. Attempt to open the file in read mode

3. Iterate over every line in the file, using the split() function on each line.

4. For each line, if the Type1 value is not currently in your dictionary, add the Type1 as a new key for the first dimension of your dictionary. Then, add the individual pokemon as a key in the second dimension, corresponding to that type. So you'll have something that looks like

{type: {name: [stat_list]}}

Computation Stage

1. For each type in the 2D dictionary, you need to create a 7-element list which contains the sum of each individual pokemon's stats. Essentially, you want something like:

[total_sum, hp_sum, atk_sum, def_sum, satk_sum, sdef_sum, spd_sum]

2. For each element in this newly generated list, you must divide by the length of the dictionary corresponding to that type.

3. This is the part where we choose if we want a speedup or not.

Speedup (memoization)

1. Create a 1D dictionary, preset to contain keys with the stat options, mapping to an empty list of 2-tuples.

{stat: [('', 0)]}

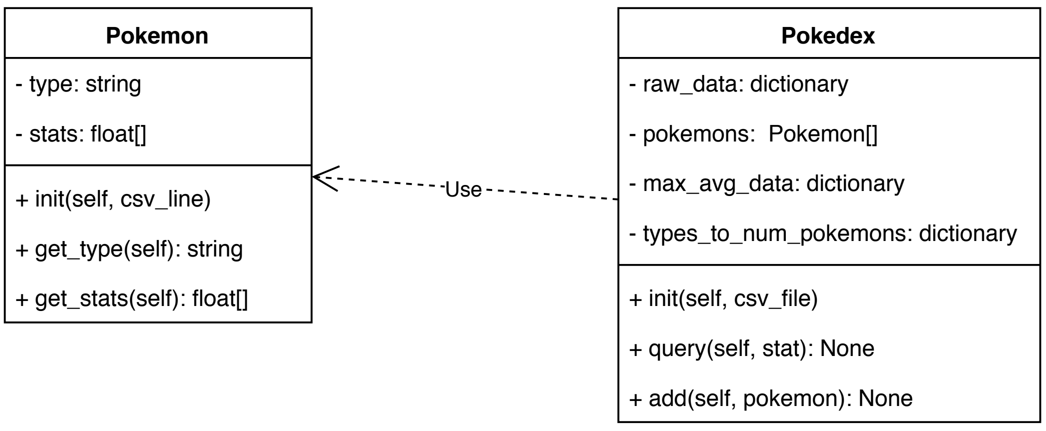
2. When you create that list of average stats for each type, compare it with the values in this dictionary. If the numeric values stored in the dictionary are less than the values in your current average list, then do a replacement.

3. Then, when accepting user queries, simply reference the values stored in this dictionary.

The Slow Way
(not memoization)

When taking queries from user input, you must repeat the Computation Stage, but storing an additional variable which holds the current maximum value for the query you're calculating. This is very slow, and very poor style, but it is, unfortunately, a valid approach for this assignment.

Possible Solution 2: The Fun Way



The above solution and diagram to be discussed in class on Monday (6/16/19). Including discussion of improvements, why you would choose an OOP style for this program, and other good notes.