CSE 4256

Due Sunday Feb 13 11:59 PM EST

Dr. Alan Weide

Spring Term 2022

Homework 4

Assignment

This assignment should be completed as a single Python source (`.py`) file modified from `hw4-template.py` (available on Carmen). Each question (including the Challenge Activities) has an associated `TODO` comment in the template file.

Argument Types. In this section, we'll explore the different argument types permitted by Python. Each function you write should simply print the name and value of every argument passed to the function by the caller.

(1) The function `printargs0` is implemented and takes a number of arguments. Write a statement that calls this function.

(2) Write a function called `printargs1` that takes four arguments, all with default values, and call it.

(3) Write a function called `printargs2` that takes one positional-only argument, two positional-or-keyword arguments, and three keyword-only arguments, and call it.

(4) Write a function called `printargs3` that takes four positional-only arguments, and call it.

(5) Write a function called `printargs4` that takes a *variadic argument list*, and call it.

(6) Write a function called `printargs5` that takes a *keyword-variadic argument*, and call it.

Dictionaries. In this section, you'll explore dictionaries and their many uses.

(7) Write a function called `build_dict1(keys, values)` that, given two lists of equal length returns a dictionary in which each item in `keys` is associated with the corresponding item in `values`. Use a for-in loop to do so.

(8) Write a function called `build_dict2(keys, values)` that has the same behavior as `build_dict1`, but *does not* use a loop. Specifically, your implementation of `build_dict2` should use a dictionary comprehension.

(9) Write a function called `build_dict3(keys, values)` that has the same behavior as `build_dict1` and `build_dict2`, but uses neither a loop nor a comprehension. Specifically, your implementation of `build_dict3` should use the built-in **`zip`**`(seq1, seq2)` function that produces a sequence consisting of pairs of corresponding items from `seq1` and `seq2`.

(10) Write a function called `letter_freq(s:` **`str`**`) -> ` **`dict`** that, given a string `s`, returns a dictionary which maps each letter to the number of times it appears in the string. The function should be case-insensitive, but the dictionary keys should be *uppercase* letters. If a letter does not appear in the string, it should not appear in the returned dictionary. **Hint**: use the functions **`str.`**`isalpha()`, **`str.`**`isupper()`, or **`str.`**`islower()` to identify letters in `s`.

(11) Write a function called `popular_letter(s:` **`str`**`) -> ` **`str`** that, given a string `s`, returns the letter in that string that appears most often. **Hint**: use the `letter_freq` function you just created as well as the built-in **`max`** function (see `https://docs.python.org/3/library/functions.html#max`).

Generators and Lambdas. In this section we'll both build and use generators and lambda expressions.

(12) Write a generator function called `collatz(x)` that generates the series identified by the Collatz Conjecture starting with value `x`. That is, it should generate the next number in the series based on the following equation (the series is sometimes called a "Hailstone Series"):

$$C_{n+1} = \begin{cases} C_n/2 & C_n \text{ is even} \\ C_n * 3 + 1 & C_n \text{ is odd} \end{cases}$$

Note: we say the series is terminated when its value is 1. Think about why that is the case.

(13) Write a function called `collatz_len(x)` that returns the length of the series generated by `collatz(x)`.

(14) Write a generator function called `words(s)` that iterates through every "word" (*i.e.*, contiguous sequence of letter characters) in a string. For example, when

`s == "Hello, this is Alan."`, the generator should produce the strings `"Hello"`, `"this"`, `"is"`, and `"Alan"` in order.

(15) Write a function called `mapped_list(lst, f)` that returns a new list that is the result of applying the single-argument function `f` to each item in `lst`.

(16) Call `mapped_list` with a lambda expression such that the created list consists of the length of the Collatz Conjecture series (as defined above in #12) generated by each element of `lst`.

Challenge Activities

Some homeworks (such as this one) will have additional challenge activities. These activities **do not contribute to your grade**, but they are problems that I find interesting or challenging.

(17) The `mcg` function is an exceptionally simple pseudorandom number generator. Play with the apparently-arbitrary values in the function. Can you make the results noticeable better or worse by chaning them?

(18) The `diceroller` function on the slides is fine, but the output leaves something to be desired. Modify the `diceroller` function so that the output is a horizontal bar graph made of ASCII characters. Do not import any charting modules—only use built-in functions for **`dict`** and **`str`**. A useful character to use as you build each bar is the Unicode character 'FULL BLOCK' (U+2588), which can be included in Python source as the **`str`** literal `u"\u2588"`. Here are the requirements for your output:

- There should be one line for each possible roll value, and they should be sorted in increasing order.
- Each line should begin with a label (*i.e.*, a roll value) followed by at least one space, followed by the bar for that value.
- Each bar should be left-aligned in the console.
- Each line should be no more than 80 characters wide, including the label.

(19) Use the generator you created in problem #14 to write a function called `firstlines(file_name)` that builds a dictionary of every word that appears in a file

mapped to the line number on which it first appears. **Hint**: to read from a file, use the built-in function **`open`**`(file_name)`, which opens the file with an iterator over the lines of that file. Each line of the file is provided as a **`str`** ending in a newline character `"\n"`.

Submission

To submit this assignment, upload your modified `hw4-template.py` file containing all of your code and question answers to the "Homework 4" assignment on Carmen. If there are test cases in your modified file, please comment them out before uploading it. As always, be sure to note all group members who contributed to the assignment and what those contributions were.