Python Code Challenges: Dictionaries (Advanced)

Difficult Python Code Challenges Involving Dictionaries

This article will help you review Python functions by providing some code challenges involving dictionaries.

Some of these challenges are difficult! Take some time to think about them before starting to code.

You might not get the solution correct on your first try — look at your output, try to find where you're going wrong, and iterate on your solution.

Finally, if you get stuck, use our solution code! If you "Check Answer" twice with an incorrect solution, you should see an option to get our solution code. However, truly investigate that solution — experiment and play with the solution code until you have a good grasp of how it is working. Good luck!

Function Syntax

```
def some_function(some_input1, some_input2):
  … do something with the inputs …
  return output
```

For example, a function that counts the number of values in a dictionary that are above a given number would look like this:

```
def greater_than_ten(my_dictionary, number):
  count = 0
  for value in my_dictionary.values():
    if value > number:
      count += 1
  return count
```

And this would produce output like:

```
>>> greater_than_ten({"a":1, "b":2, "c":3}, 0)
3
>>> greater_than_ten({"a":1, "b":2, "c":3}, 5)
0
```

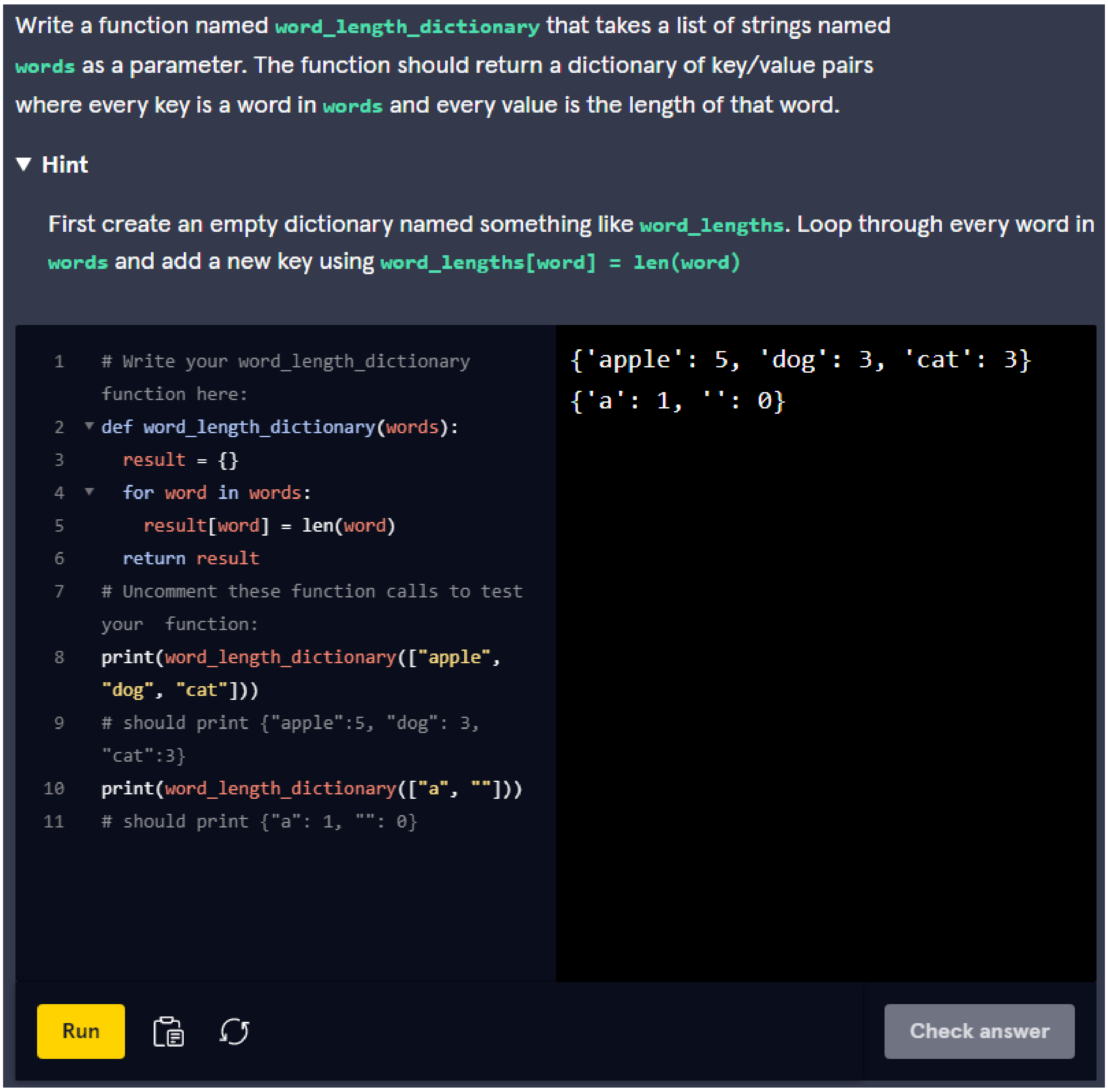
Challenges

We've included 4 challenges below. Try to answer all of them and polish up your problem-solving skills!

1. Word Length Dict

Let's start by writing a function that creates a new dictionary based on a list of strings. The keys of our dictionary will be every string in our list of strings, while the values will be the length of each of the words in the string list. Here are the steps:

1. Define the function to accept one parameter for our list of strings
2. Create a new empty dictionary
3. Loop through every string in the list of strings
4. Inside the loop, add the string as a key and the length of the string as the value to the dictionary
5. After the loop, return the new dictionary



Write a function named `word_length_dictionary` that takes a list of strings named `words` as a parameter. The function should return a dictionary of key/value pairs where every key is a word in `words` and every value is the length of that word.

First create an empty dictionary named something like `word_lengths`. Loop through every word in `words` and add a new key using `word_lengths[word] = len(word)`

```python
# Write your word_length_dictionary function here:
def word_length_dictionary(words):
  result = {}
  for word in words:
    result[word] = len(word)
  return result
# Uncomment these function calls to test your  function:
print(word_length_dictionary(["apple", "dog", "cat"]))
# should print {"apple":5, "dog": 3, "cat":3}
print(word_length_dictionary(["a", ""]))
# should print {"a": 1, "": 0}
```

```
{'apple': 5, 'dog': 3, 'cat': 3}
{'a': 1, '': 0}
```

Here is this solution:

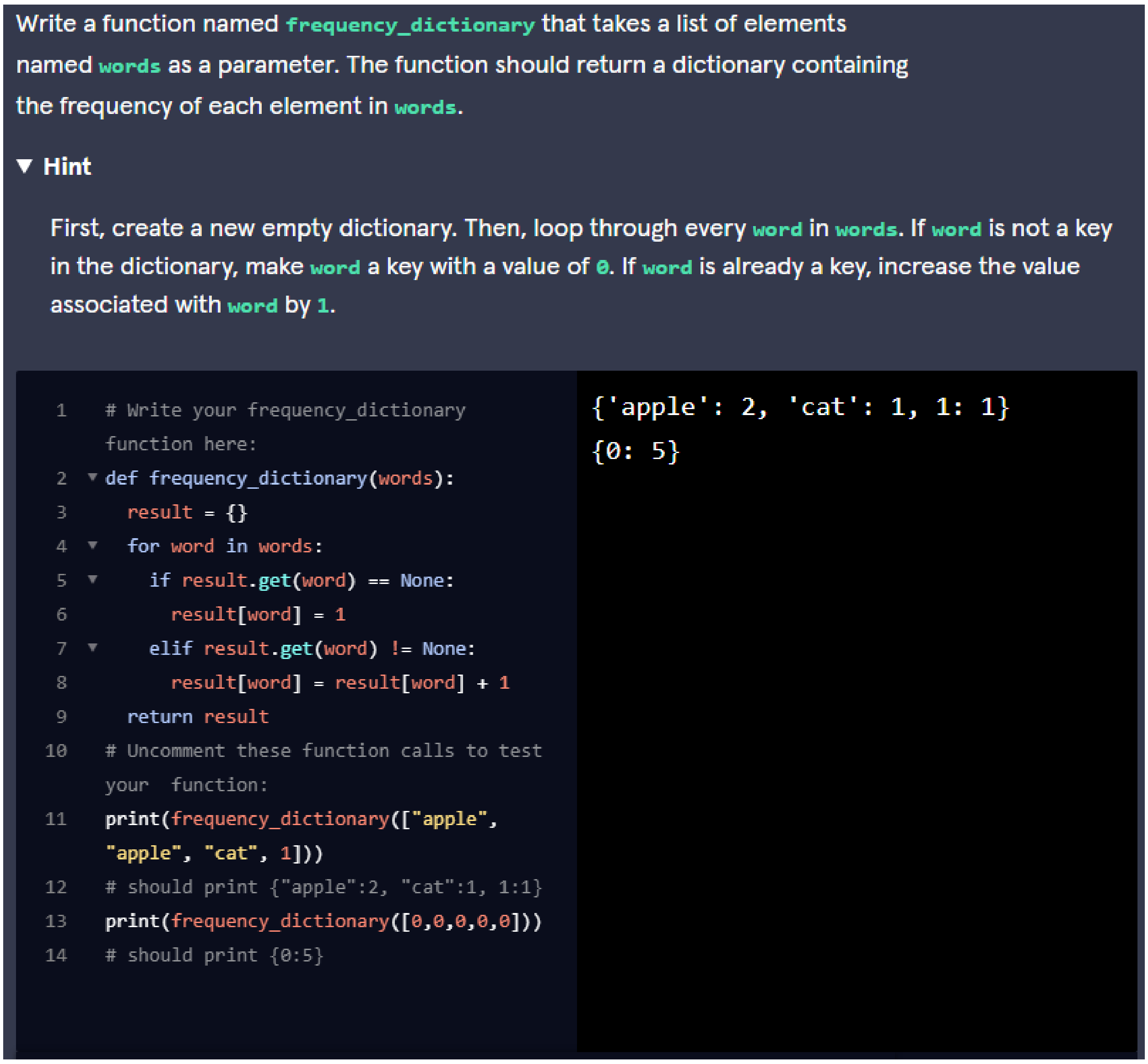
```
def word_length_dictionary(words):
  word_lengths = {}
  for word in words:
    word_lengths[word] = len(word)
  return word_lengths
```

To create a new dictionary we set a variable equal to `{}`. While iterating through each string in our string list, we can add the key and value to our dictionary using this syntax: `word_lengths[word] = len(word)`.

2. Frequency Count

This next function is similar, but instead of counting the length of each string in the list of strings, we will be counting the frequency of each string. This function will also accept a list of strings as input and return a new dictionary. Here is what we need to do:

1. Define the function to accept one parameter for our list of strings
2. Create a new empty dictionary
3. Loop through every string in the list of strings
4. Inside the loop, if the string is not already in our dictionary, store the string as a key with a value of 0 in our dictionary. Then, increment the value by 1.
5. After the loop, return the new dictionary



Write a function named `frequency_dictionary` that takes a list of elements named `words` as a parameter. The function should return a dictionary containing the frequency of each element in `words`.

▼ **Hint**

First, create a new empty dictionary. Then, loop through every `word` in `words`. If `word` is not a key in the dictionary, make `word` a key with a value of `0`. If `word` is already a key, increase the value associated with `word` by `1`.

```python
# Write your frequency_dictionary function here:
def frequency_dictionary(words):
  result = {}
  for word in words:
    if result.get(word) == None:
      result[word] = 1
    elif result.get(word) != None:
      result[word] = result[word] + 1
  return result
# Uncomment these function calls to test your  function:
print(frequency_dictionary(["apple", "apple", "cat", 1]))
# should print {"apple":2, "cat":1, 1:1}
print(frequency_dictionary([0,0,0,0,0]))
# should print {0:5}
```

```
{'apple': 2, 'cat': 1, 1: 1}
{0: 5}
```

Here is how we solved it:

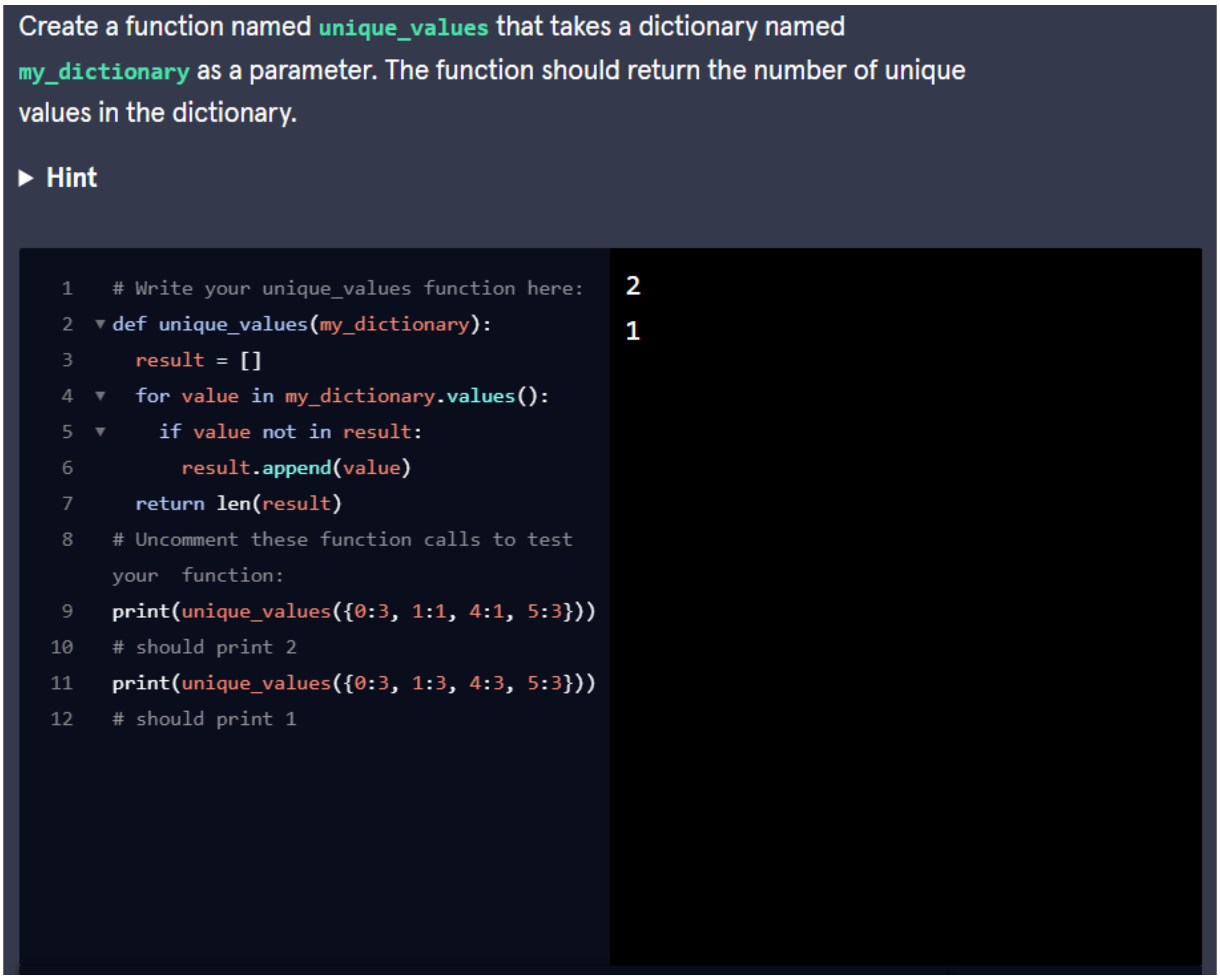
```python
def frequency_dictionary(words):
  freqs = {}
  for word in words:
    if word not in freqs:
      freqs[word] = 0
    freqs[word] += 1
  return freqs
```

To create a new dictionary we set a variable equal to `{}`. We iterate through each of the strings in the list of strings and check if it is already in our dictionary using the `in` keyword. If it is not then we add it as a new key-value pair where the value is 0. Regardless of whether the string was already in the dictionary, increase the value by 1. This will make it so all new entries will have a value of 1 and all existing entries will increase their old value by 1.

3. Unique Values

Now let's try reading a dictionary as input and finding how many values are unique. The function should return a number which is the count of all values from the dictionary without including duplicates. These are the steps:

1. Define the function to accept one parameter for our dictionary
2. Create a new empty list
3. Loop through every value in our dictionary
4. Inside the loop, if the value is not already in our list, append the value to our list
5. After the loop, return the length of our list



Create a function named `unique_values` that takes a dictionary named `my_dictionary` as a parameter. The function should return the number of unique values in the dictionary.

▶ Hint

```python
# Write your unique_values function here:
def unique_values(my_dictionary):
  result = []
  for value in my_dictionary.values():
    if value not in result:
      result.append(value)
  return len(result)
# Uncomment these function calls to test your  function:
print(unique_values({0:3, 1:1, 4:1, 5:3}))
# should print 2
print(unique_values({0:3, 1:3, 4:3, 5:3}))
# should print 1
```

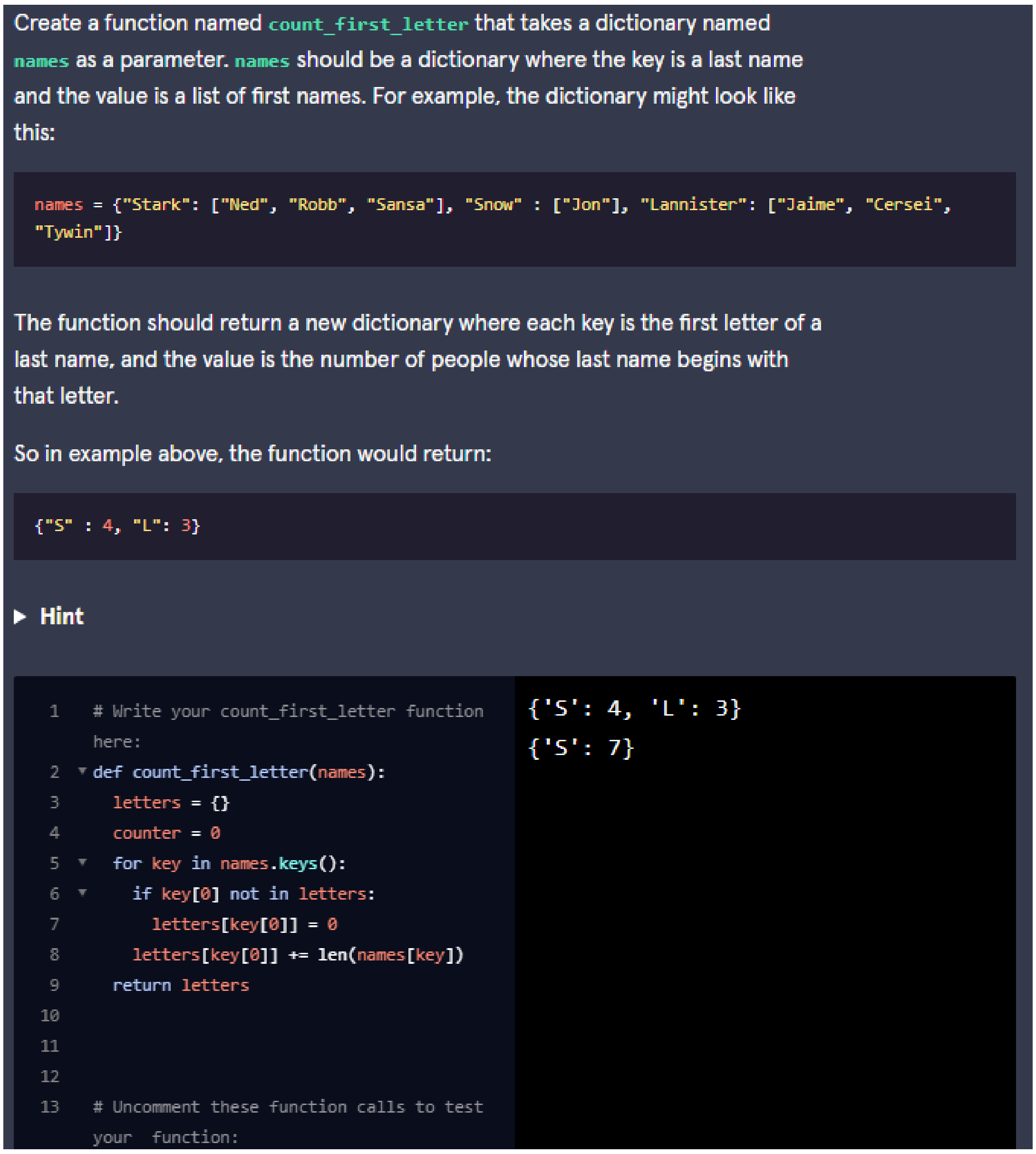
```
2
1
```

This function has a similar structure to the last one except that the input has been changed to a dictionary. We iterate through each of the values and whenever we find one we have not added to our list already, we add it to the list. After the loop, we return the length of the list since that contains all unique values from the dictionary.

4. Count First Letter

This function accepts a dictionary where the keys are last names and the values are lists of first names of people who have that last name. We need to calculate the number of people who have the same first letter in their last name. Here are the steps we need:

1. Define the function to accept one parameter for our dictionary
2. Create a new empty dictionary called `letters`
3. Loop through every key in our `names` dictionary
4. Inside the loop, get the first letter of the last name we are looking at. If the first letter is not in our letter dictionary, add it as a key with a value of 0. Then, increment that key by the number of people that have that last name
5. After the loop, return the `letters` dictionary



Create a function named `count_first_letter` that takes a dictionary named `names` as a parameter. `names` should be a dictionary where the key is a last name and the value is a list of first names. For example, the dictionary might look like this:

```
names = {"Stark": ["Ned", "Robb", "Sansa"], "Snow" : ["Jon"], "Lannister": ["Jaime", "Cersei", "Tywin"]}
```

The function should return a new dictionary where each key is the first letter of a last name, and the value is the number of people whose last name begins with that letter.

So in example above, the function would return:

```
{"S" : 4, "L": 3}
```

▶ Hint

```
# Write your count_first_letter function here:
def count_first_letter(names):
  letters = {}
  counter = 0
  for key in names.keys():
    if key[0] not in letters:
      letters[key[0]] = 0
    letters[key[0]] += len(names[key])
  return letters



# Uncomment these function calls to test your function:
```

```
{'S': 4, 'L': 3}
{'S': 7}
```

Here is what we did:

```
def count_first_letter(names):
  letters = {}
  for key in names:
    first_letter = key[0]
    if first_letter not in letters:
      letters[first_letter] = 0
    letters[first_letter] += len(names[key])
  return letters
```

This function uses two dictionaries instead of one dictionary and one list. We iterate through each of the keys (which are the last names) and store the first letter of the last name in `first_letter`. We then use similar logic to what we have used before by testing if we have already seen that letter before. If we haven't seen that letter before, then we add it to our dictionary with a value of 0. Next, we are going to increment the value. Since we know that some people share the last name (as seen by the list of first names in our `names` dictionary), we are going to increment the value in our `letters` dictionary by the length of first names that share the last name for our current iteration (`key`).