**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

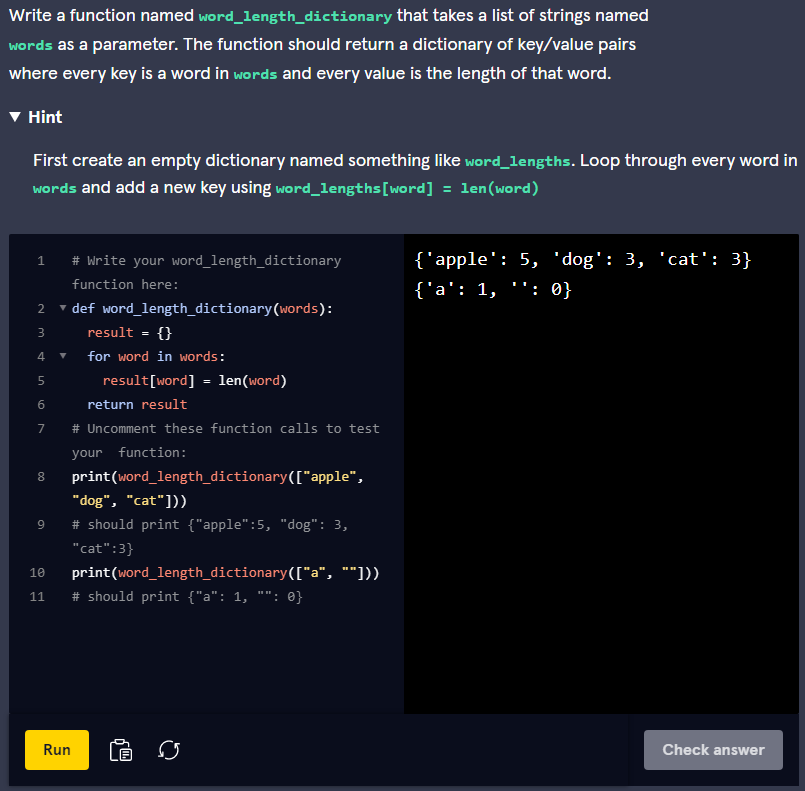
**Challenges**

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



Here is this solution:

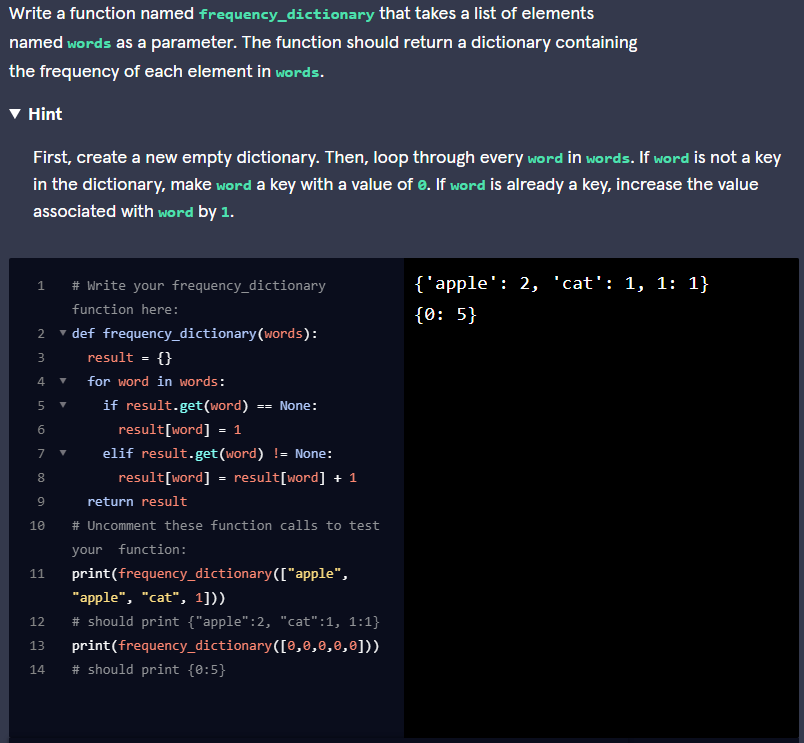
def word\_length\_dictionary(words):  
  word\_lengths = {}  
  for word in words:  
    word\_lengths[word] = len(word)  
  return word\_lengths

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



Here is how we solved it:

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.