**Python Code Challenges: Dictionaries**

**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

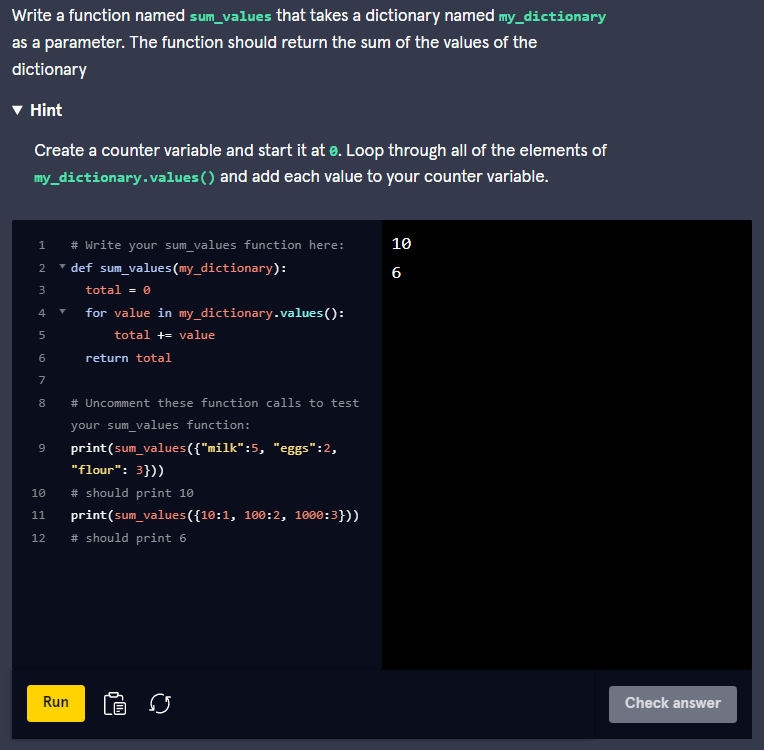
**Challenges**

We’ve included 5 challenges below. Try to answer all of them and polish up your problem-solving skills!

**1. Sum Values**

For the first code challenge, we are going to look at only the values in a dictionary. This function should sum up all of the values from the key-value pairs in the dictionary. Here are the steps we need:

1. Define the function to accept one parameter for our dictionary
2. Create a variable to keep track of our sum
3. Loop through every value in the dictionary
4. Inside the loop, add each value to the sum
5. Return the sum



Here is this solution:

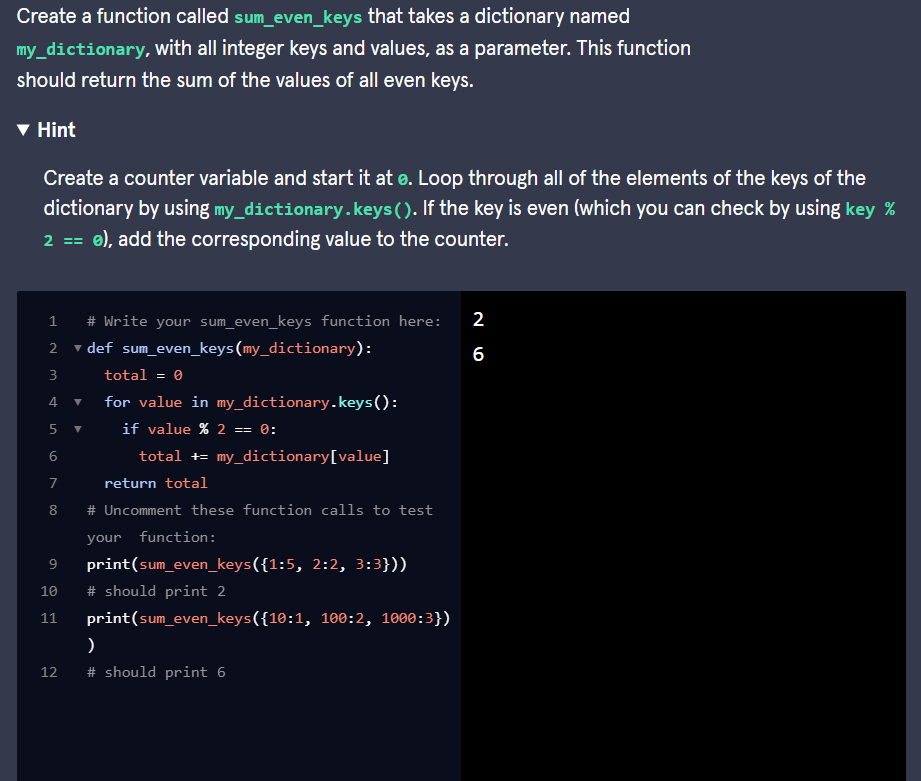
def sum\_values(my\_dictionary):  
  total = 0  
  for value in my\_dictionary.values():  
    total += value  
  return total

We start by creating a variable to keep track of the total. Next, we use the **values()** function in our **for** loop in order to iterate through each of the values in the dictionary. Using this, we can access each value and add it to our **total** variable. At the end of our loop, we return the **total**.

### 2. Even Keys

Next, we are going to do something similar, but we are going to use the keys in order to retrieve the values. Additionally, we are going to only look at every even key within the dictionary. Here are the steps:

1. Define the function to accept one parameter for our dictionary
2. Create a variable to keep track of our sum
3. Loop through every key in the dictionary
4. Inside the loop, if the key is even, add the value from the even key
5. After the loop, return the sum



Here is one solution:

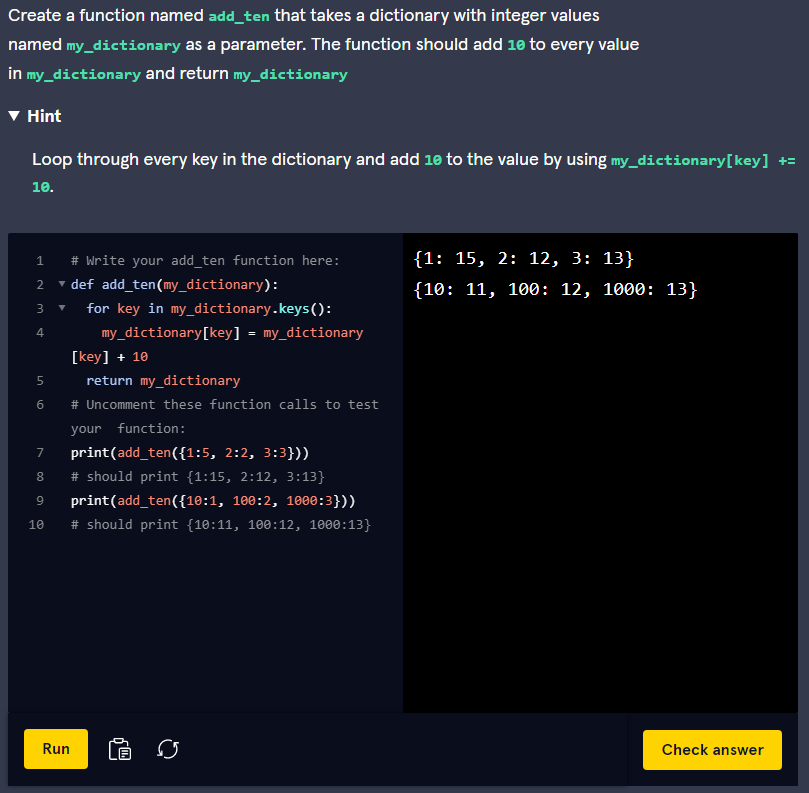
def sum\_even\_keys(my\_dictionary):  
  total = 0  
  for key in my\_dictionary.keys():  
    if key%2 == 0:  
      total += my\_dictionary[key]  
  return total

Similar to the previous problem, we are iterating through our dictionary, except this time we are iterating through the keys instead of the values. In order to get the keys we use the **keys()** function and to get the value of a key we can use brackets. To test if the key is even we use the modulus operator and test if the remainder is 0 when dividing by 2.

### 3. Add Ten

Let’s loop through the keys again, but this time let’s modify the values within the dictionary. Our function should add 10 to every value in the dictionary and return the modified dictionary. Here is what we need to do:

1. Define the function to accept one parameter for our dictionary
2. Loop through every key in the dictionary
3. Retrieve the value using the key and add 10 to it. Make sure to re-save the new value to the original key.
4. After the loop, return the modified dictionary



Here is how we did it:

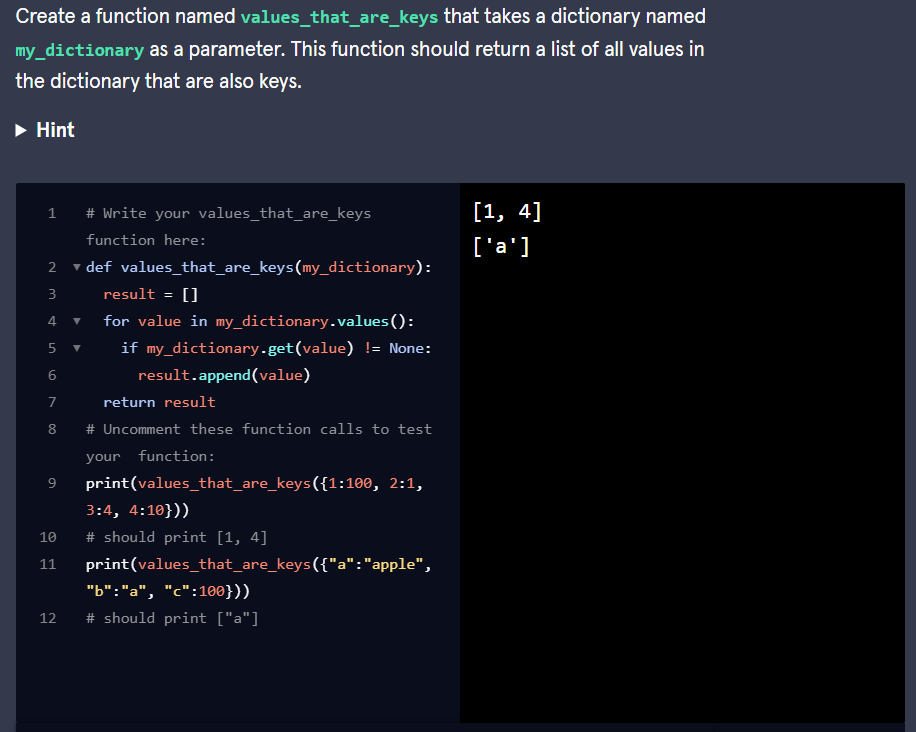
def add\_ten(my\_dictionary):  
  for key in my\_dictionary.keys():  
    my\_dictionary[key] += 10  
  return my\_dictionary

We use a **for** loop to iterate through each key and we access the value using the key. After accessing it, we overwrite the value with the value plus 10. Finally, we return the modified dictionary.

### 4. Values That Are Keys

We are making a program that will create a family tree. Using a dictionary, we want to return a list of all the children who are also parents of other children. Using dictionaries we can consider those people to be values which are also keys in our dictionary of family data. Here is what we need to do:

1. Define the function to accept one parameter for our dictionary
2. Create an empty list to hold the values we find
3. Loop through every value in the dictionary
4. Inside the loop, test if the current value is a key in the dictionary. If it is then append it to the list of values we found
5. After the loop, return the list of values which are also keys



Here is this solution:

def values\_that\_are\_keys(my\_dictionary):  
  value\_keys = []  
  for value in my\_dictionary.values():  
    if value in my\_dictionary:  
      value\_keys.append(value)  
  return value\_keys

For this solution, we iterate through every value within the dictionary. In order to check if it is also a key, we can use the **in** keyword. This checks the value against all of the keys in the dictionary to see if it exists as a key as well. If it does, then we append it to our list of values which are also keys.