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
```

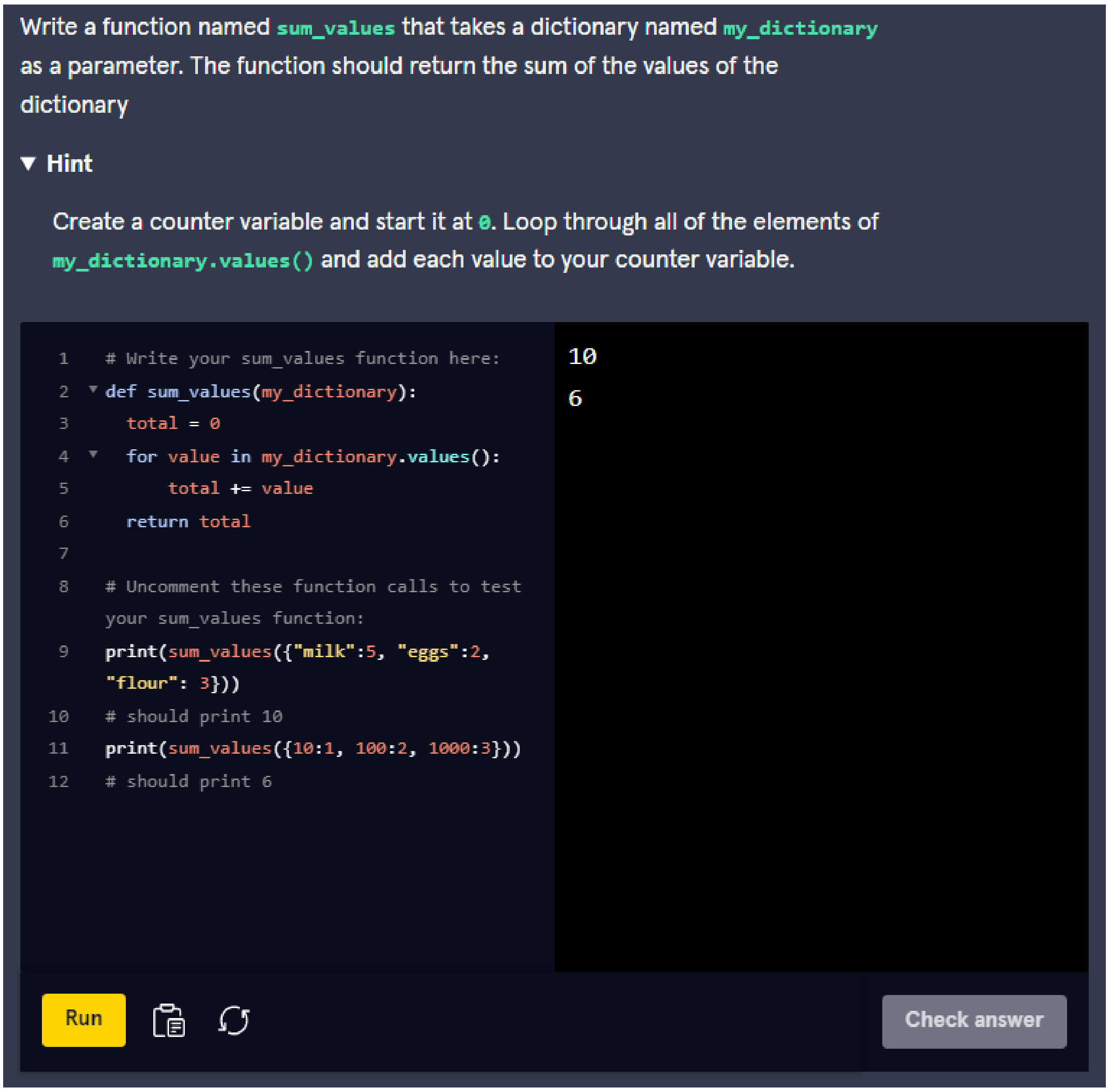
Challenges

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



Write a function named `sum_values` that takes a dictionary named `my_dictionary` as a parameter. The function should return the sum of the values of the dictionary

▼ Hint

Create a counter variable and start it at `0`. Loop through all of the elements of `my_dictionary.values()` and add each value to your counter variable.

```python
# Write your sum_values function here:
def sum_values(my_dictionary):
  total = 0
  for value in my_dictionary.values():
      total += value
  return total

# Uncomment these function calls to test your sum_values function:
print(sum_values({"milk":5, "eggs":2, "flour": 3}))
# should print 10
print(sum_values({10:1, 100:2, 1000:3}))
# should print 6
```

```
10
6
```

Here is this solution:

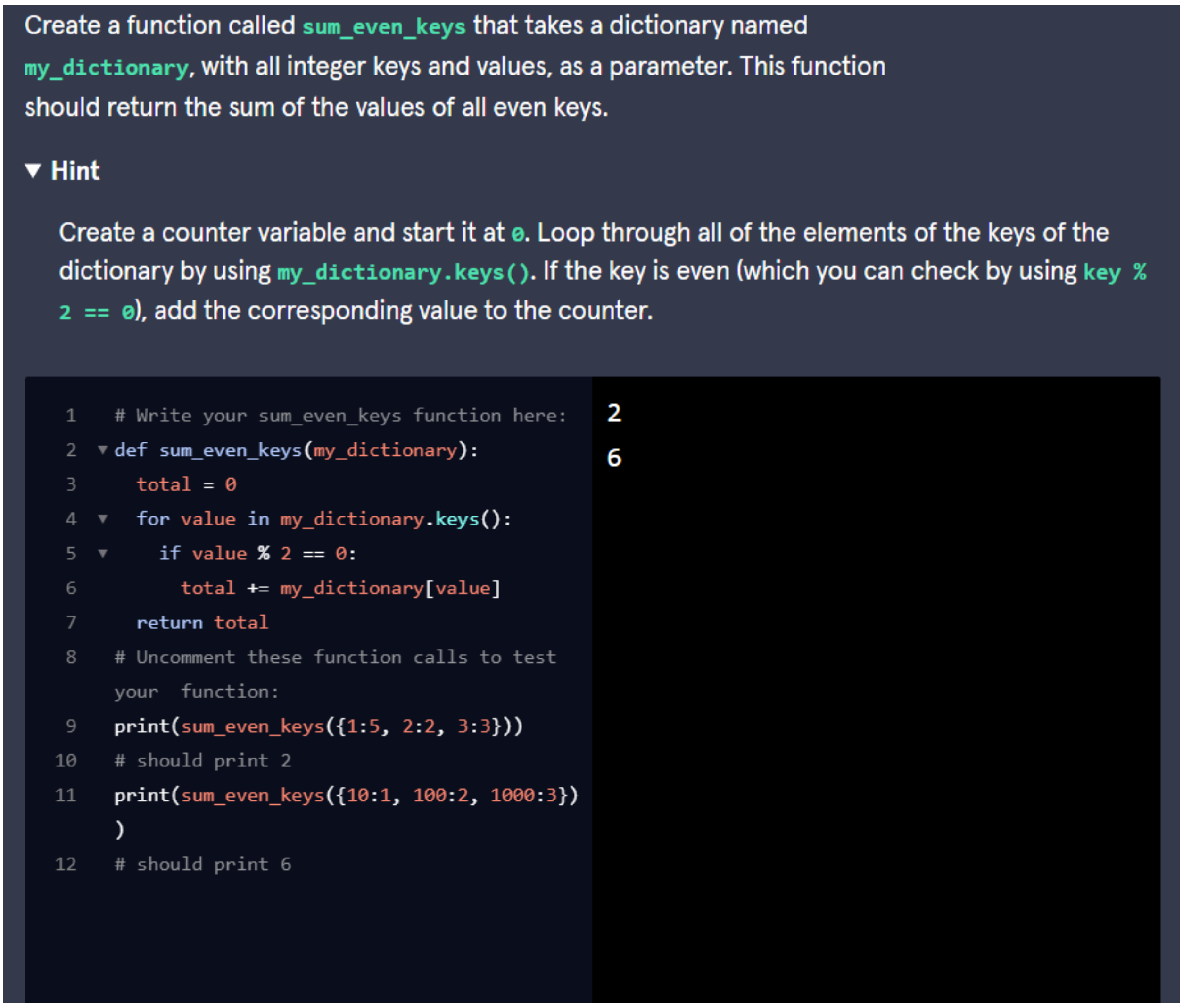
```python
def sum_values(my_dictionary):
  total = 0
  for value in my_dictionary.values():
    total += value
  return total
```

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



Create a function called `sum_even_keys` that takes a dictionary named `my_dictionary`, with all integer keys and values, as a parameter. This function should return the sum of the values of all even keys.

▼ Hint

Create a counter variable and start it at `0`. Loop through all of the elements of the keys of the dictionary by using `my_dictionary.keys()`. If the key is even (which you can check by using `key % 2 == 0`), add the corresponding value to the counter.

```
# Write your sum_even_keys function here:
def sum_even_keys(my_dictionary):
  total = 0
  for value in my_dictionary.keys():
    if value % 2 == 0:
      total += my_dictionary[value]
  return total
# Uncomment these function calls to test your  function:
print(sum_even_keys({1:5, 2:2, 3:3}))
# should print 2
print(sum_even_keys({10:1, 100:2, 1000:3}))
# should print 6
```

```
2
6
```

Here is one solution:

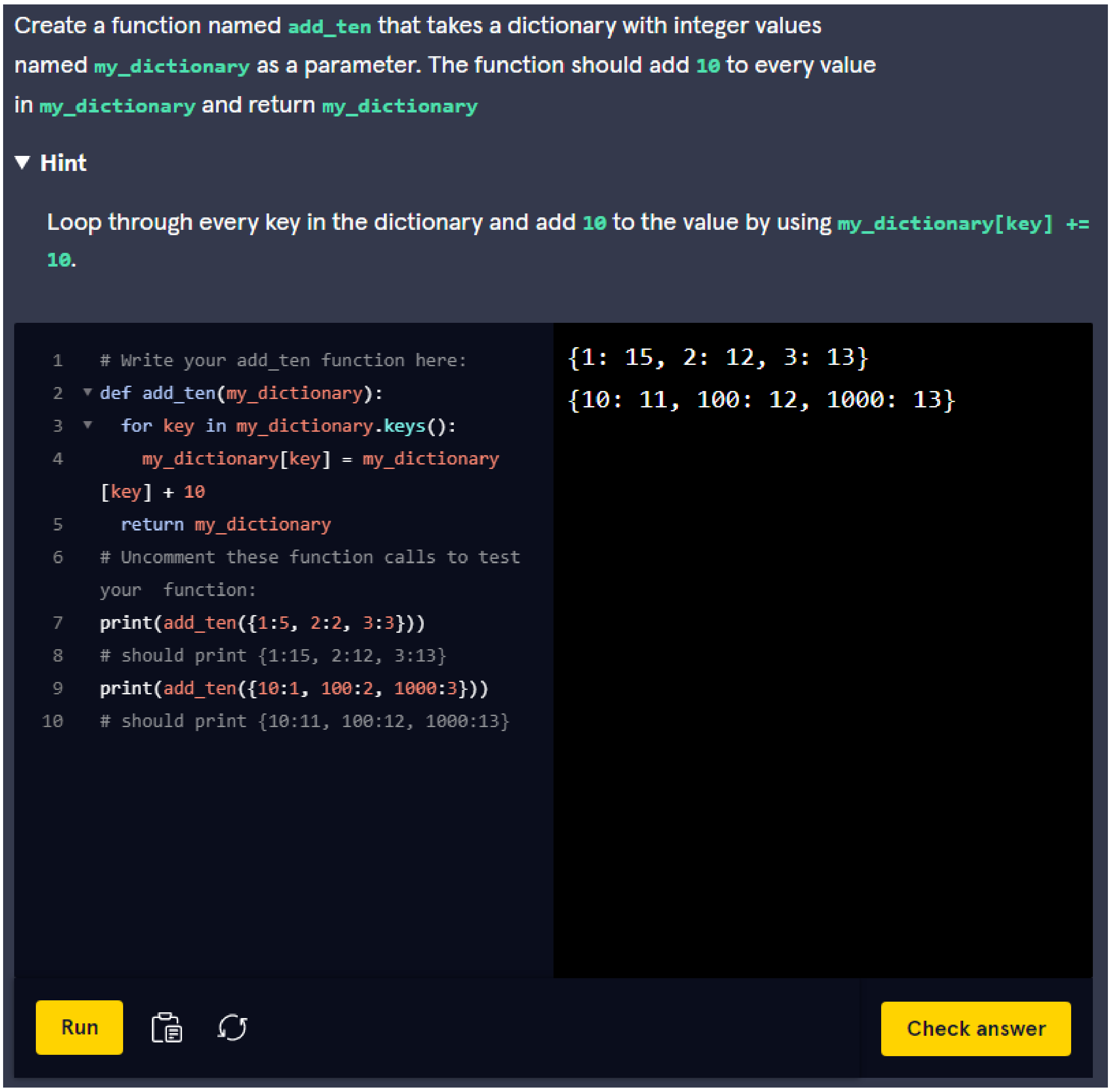
```
def sum_even_keys(my_dictionary):
  total = 0
  for key in my_dictionary.keys():
    if key%2 == 0:
      total += my_dictionary[key]
  return total
```

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


Create a function named `add_ten` that takes a dictionary with integer values named `my_dictionary` as a parameter. The function should add `10` to every value in `my_dictionary` and return `my_dictionary`

▼ **Hint**

Loop through every key in the dictionary and add `10` to the value by using `my_dictionary[key] += 10`.

```python
# Write your add_ten function here:
def add_ten(my_dictionary):
  for key in my_dictionary.keys():
    my_dictionary[key] = my_dictionary[key] + 10
  return my_dictionary
# Uncomment these function calls to test your  function:
print(add_ten({1:5, 2:2, 3:3}))
# should print {1:15, 2:12, 3:13}
print(add_ten({10:1, 100:2, 1000:3}))
# should print {10:11, 100:12, 1000:13}
```

```
{1: 15, 2: 12, 3: 13}
{10: 11, 100: 12, 1000: 13}
```

Here is how we did it:

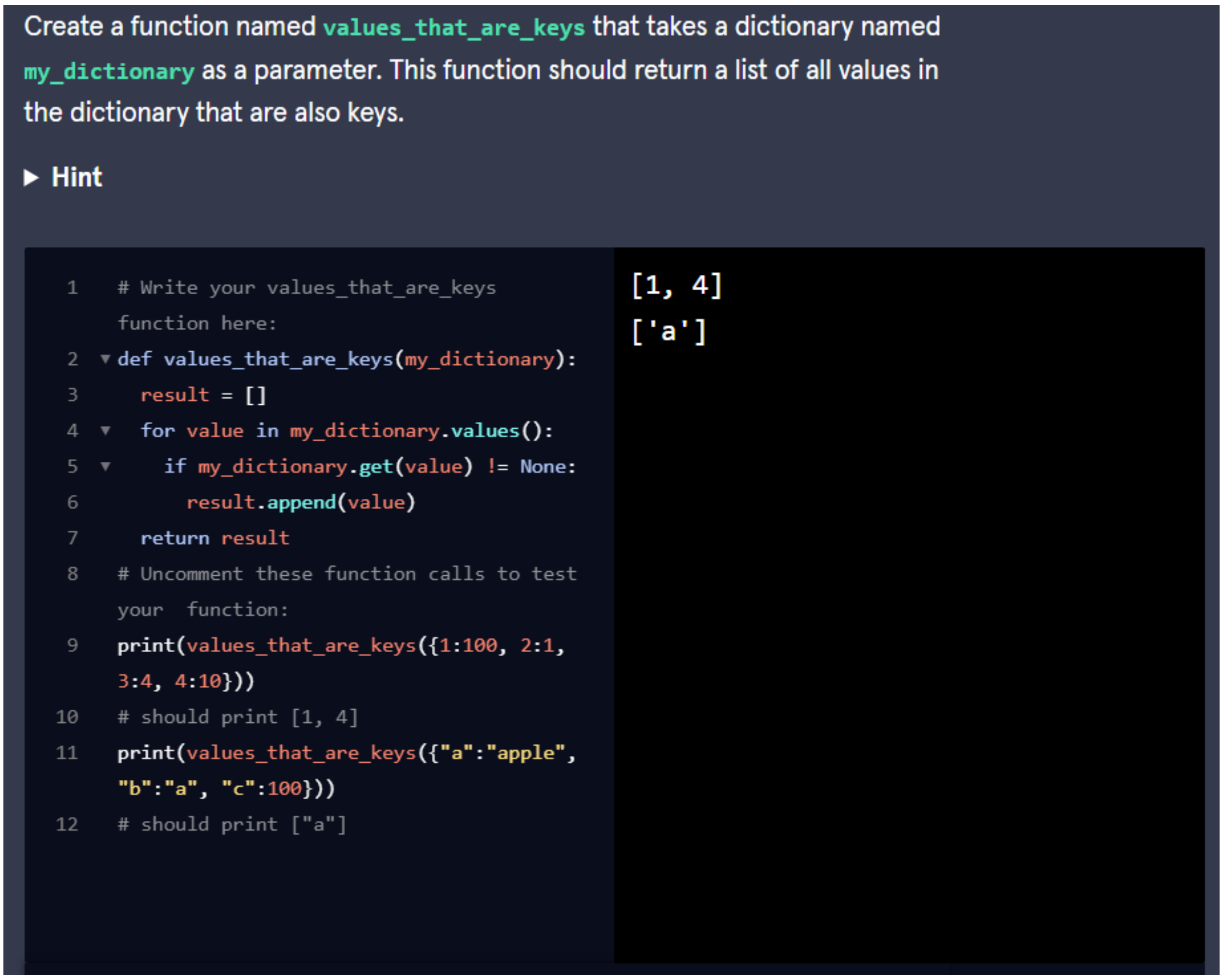
```python
def add_ten(my_dictionary):
  for key in my_dictionary.keys():
    my_dictionary[key] += 10
  return my_dictionary
```

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



Create a function named `values_that_are_keys` that takes a dictionary named `my_dictionary` as a parameter. This function should return a list of all values in the dictionary that are also keys.

▶ Hint

```
# Write your values_that_are_keys function here:
def values_that_are_keys(my_dictionary):
  result = []
  for value in my_dictionary.values():
    if my_dictionary.get(value) != None:
      result.append(value)
  return result
# Uncomment these function calls to test your  function:
print(values_that_are_keys({1:100, 2:1, 3:4, 4:10}))
# should print [1, 4]
print(values_that_are_keys({"a":"apple", "b":"a", "c":100}))
# should print ["a"]
```

```
[1, 4]
['a']
```

Here is this solution:

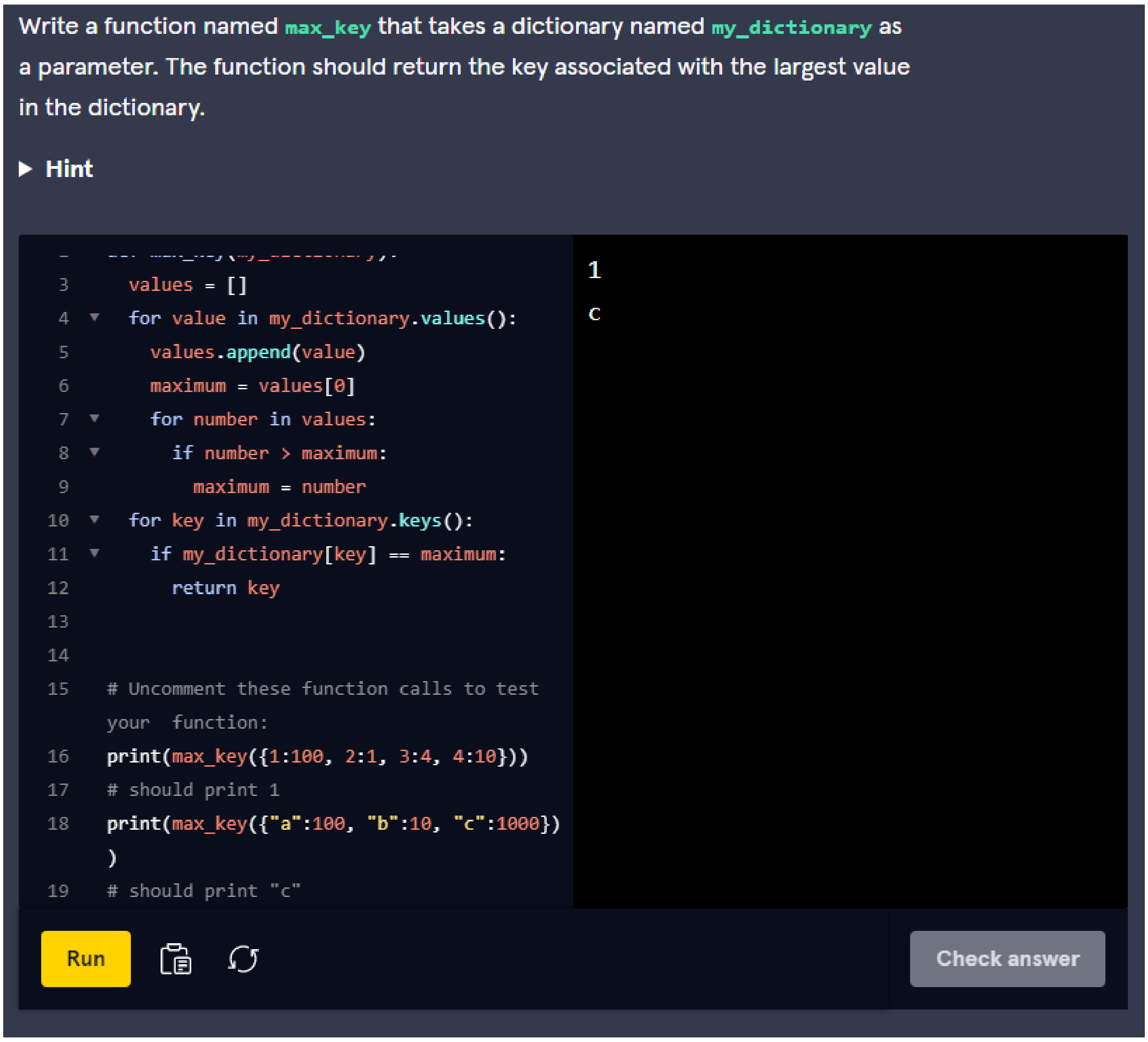
```
def values_that_are_keys(my_dictionary):
  value_keys = []
  for value in my_dictionary.values():
    if value in my_dictionary:
      value_keys.append(value)
  return value_keys
```

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.

5. Largest Value

For the last challenge, we are going to create a function that is able to find the maximum value in the dictionary and return the associated key. This is a twist on the max algorithm since it is using a dictionary rather than a list. These are the steps:

1. Define the function to accept one parameter for our dictionary
2. Initialize the starting key to a very low number
3. Initialize the starting value to a very low number
4. Iterate through the dictionary's key/value pairs.
5. Inside the loop, if the current value is larger than the current largest value, replace the largest key and largest value with the current ones you are looking at
6. Once you are done iterating through all key/value pairs, return the key which has the largest value



Write a function named `max_key` that takes a dictionary named `my_dictionary` as a parameter. The function should return the key associated with the largest value in the dictionary.

▶ **Hint**

```python
def max_key(my_dictionary):
  values = []
  for value in my_dictionary.values():
    values.append(value)
    maximum = values[0]
    for number in values:
      if number > maximum:
        maximum = number
  for key in my_dictionary.keys():
    if my_dictionary[key] == maximum:
      return key


# Uncomment these function calls to test your  function:
print(max_key({1:100, 2:1, 3:4, 4:10}))
# should print 1
print(max_key({"a":100, "b":10, "c":1000}))
# should print "c"
```

```
1
c
```

Here is this solution:

```
def max_key(my_dictionary):
  largest_key = float("-inf")
  largest_value = float("-inf")
  for key, value in my_dictionary.items():
    if value > largest_value:
      largest_value = value
      largest_key = key
  return largest_key
```

In order to program the max algorithm using dictionaries, we need to keep track of the max value and the key which is used to access it. We start by using `float("-inf")` in order to initialize them to the lowest possible value. To retrieve the key and value at the same time, we use the `items()` function. Inside our loop, we overwrite the current largest value and the key used to access whenever we find a larger value. We return the largest value’s key once we have iterated through the entire dictionary.