Python Code Challenges: Strings

Python Code Challenges involving Strings

This article will help you review Python functions by providing some code challenges about strings.

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

As a refresher, function syntax looks like this:

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

For example, a function that finds the difference in length between two Strings would look like this:

```
def lengthDiff(str1, str2):
  return len(str1) - len(str2)
```

And this would produce output like:

```
>>> lengthDiff("Python", "rocks")
1
>>> lengthDiff("Marco", "Polo")
1
>>> lengthDiff("Kevin", "Durant")
-1
```

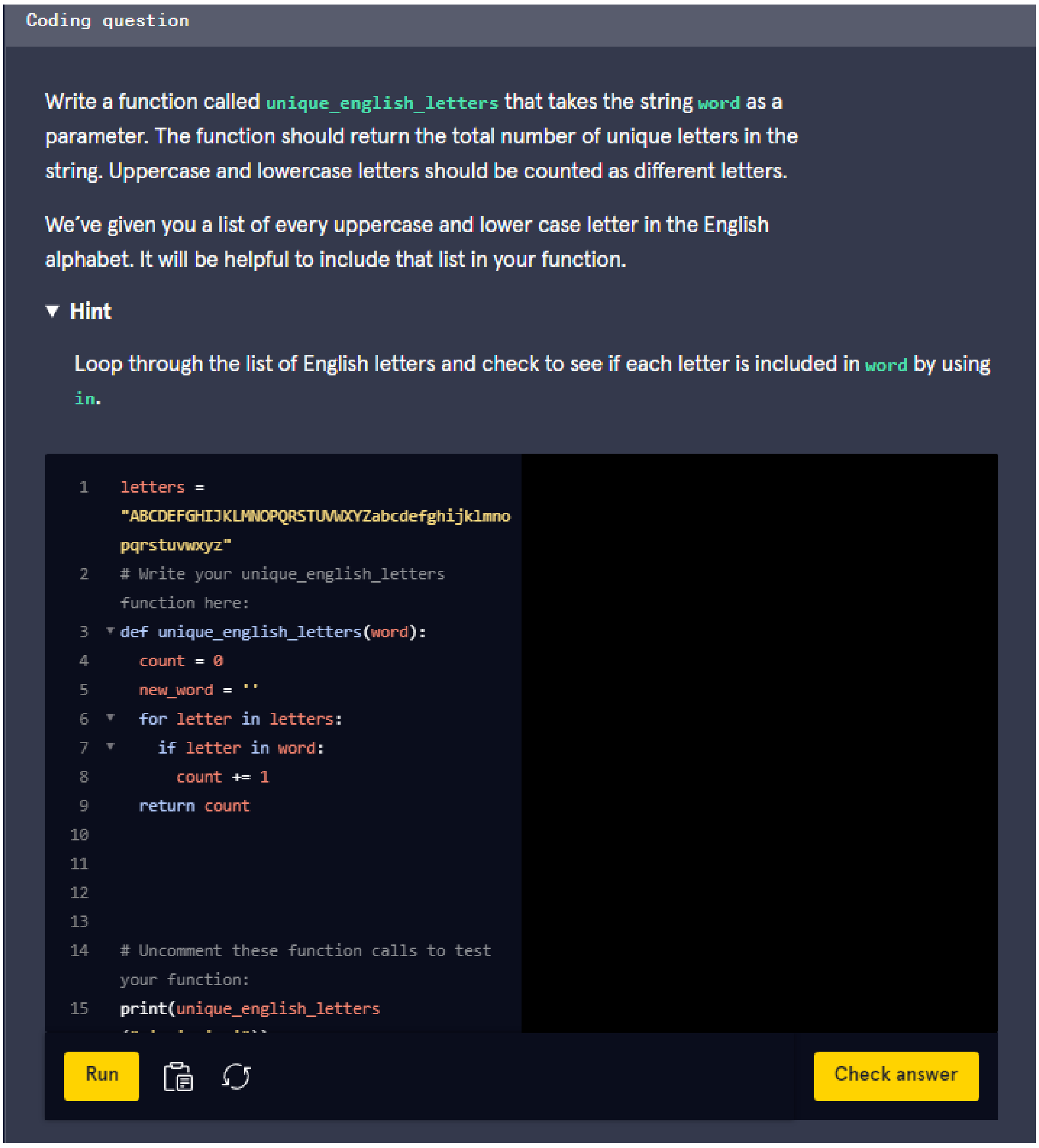
Challenges

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

1. Count Letters

For the first code challenge, we are going to count the number of unique letters in a string. This means that when we are looking at the word, any new letters should be counted and any duplicates should not be counted. There are a few ways to solve this, but we can use the provided alphabet string to ensure that duplicates are not counted. Here is what we need to do:

1. Define the function to accept one parameter — the word whose letters we are counting
2. Create a counter to keep track of unique letters
3. Loop through every letter in our alphabet string. If the current letter was found in our word, increase our count
4. Return the count after looping through all letters.



Coding question

Write a function called `unique_english_letters` that takes the string `word` as a parameter. The function should return the total number of unique letters in the string. Uppercase and lowercase letters should be counted as different letters.

We've given you a list of every uppercase and lower case letter in the English alphabet. It will be helpful to include that list in your function.

▼ **Hint**

Loop through the list of English letters and check to see if each letter is included in `word` by using `in`.

```
letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
# Write your unique_english_letters function here:
def unique_english_letters(word):
  count = 0
  new_word = ''
  for letter in letters:
    if letter in word:
      count += 1
  return count




# Uncomment these function calls to test your function:
print(unique_english_letters
```

This is how we solved it:

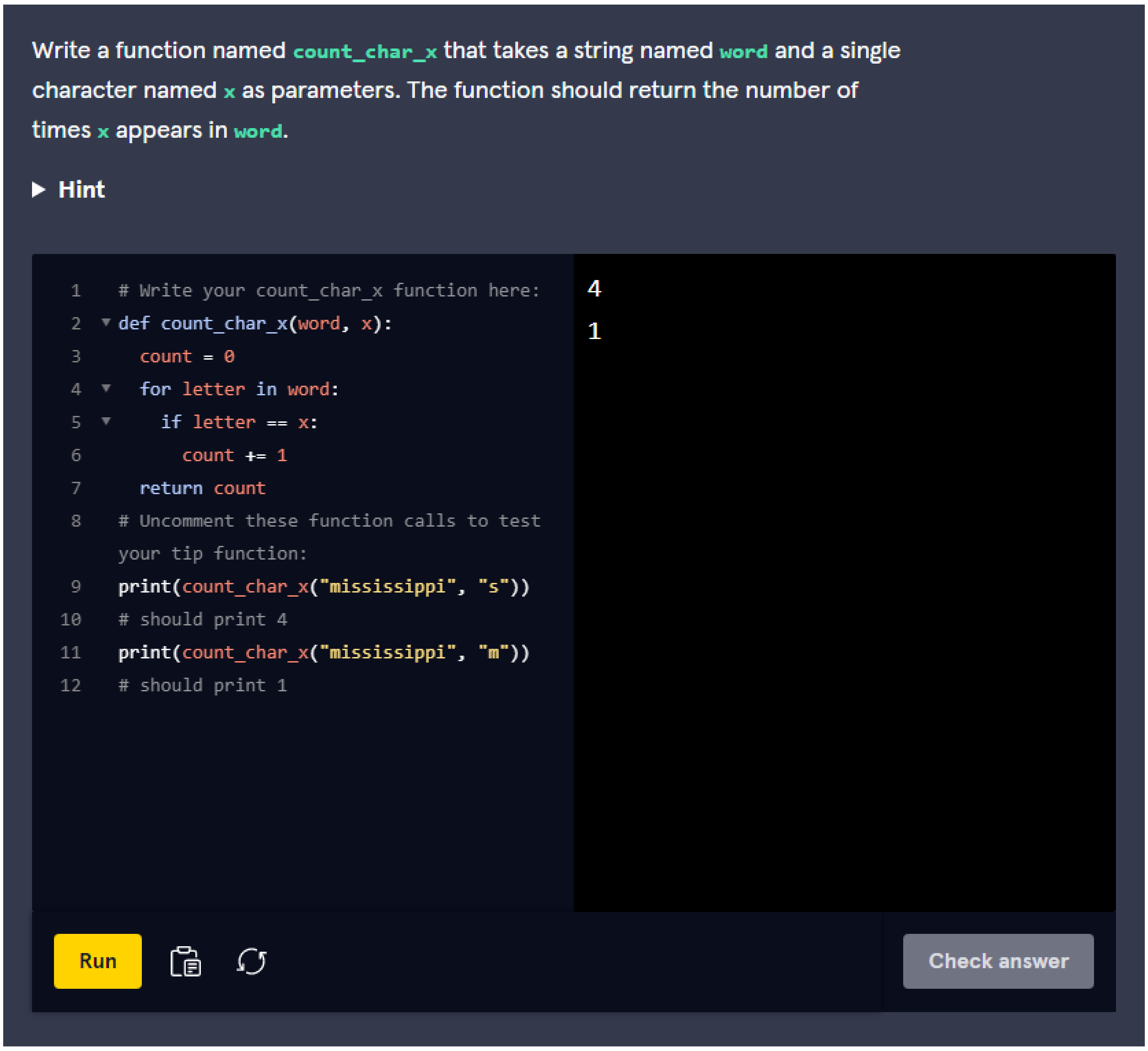
```
def unique_english_letters(word):
  uniques = 0
  for letter in letters:
    if letter in word:
      uniques += 1
  return uniques
```

Since the provided alphabet string includes a single instance of all uppercase and lowercase letters in the English alphabet, we can iterate through that string and see if our input strings contains the current letter we are looking at. This can be accomplished using the keyword `in`. For every letter `in` the possible letters, we see if that letter is `in` our string!

2. Count X

Next, we are going to try something a bit different. This time we are going to count the number of occurrences of a certain letter within a string. Our function will accept a parameter for a string and another for the character which we are going to count. For example, providing the word `"mississippi"` and the character `'s'` would result in an answer of 4. These are the steps we need to take:

1. Define the function to accept two parameters. `word` for the input string and `x` for the single character
2. Create a counter to keep track of the occurrences
3. Loop through every letter in the string. If the current letter is equal to the input letter, increase our counter
4. Return the counter after looping through the entire string.


Write a function named `count_char_x` that takes a string named `word` and a single character named `x` as parameters. The function should return the number of times `x` appears in `word`.

▶ Hint

```
# Write your count_char_x function here:
def count_char_x(word, x):
  count = 0
  for letter in word:
    if letter == x:
      count += 1
  return count
# Uncomment these function calls to test your tip function:
print(count_char_x("mississippi", "s"))
# should print 4
print(count_char_x("mississippi", "m"))
# should print 1
```

```
4
1
```

Here is one way to solve it:

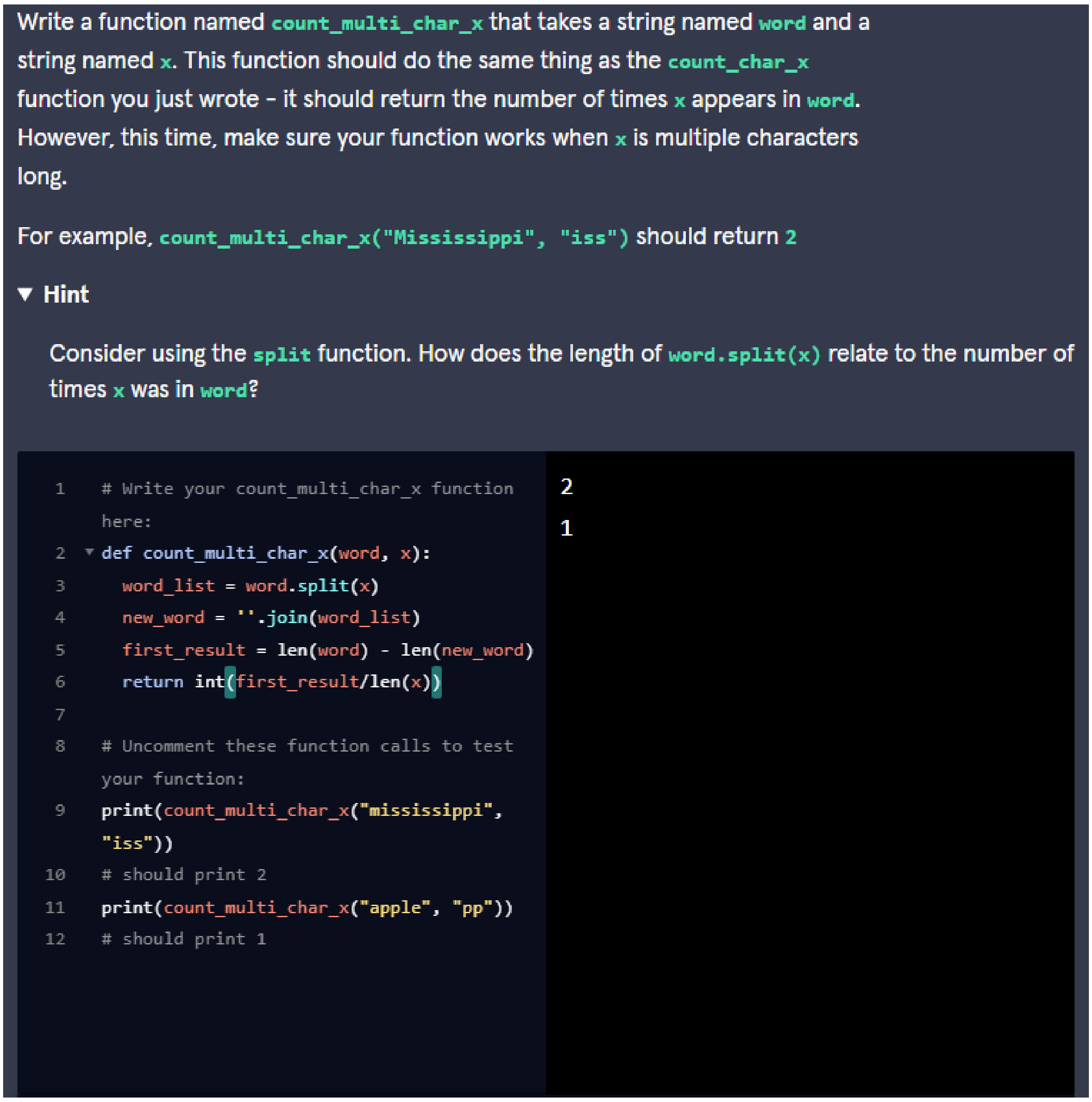
```
def count_char_x(word, x):
  occurrences = 0
  for letter in word:
    if letter == x:
      occurrences += 1
  return occurrences
```

This solution is similar to the last solution. In this case, we are looping through the input string and comparing it against the input character. If they are the same, then we increase the counter.

3. Count Multi X

Now let's change our function to compare against an entire string instead of a single character. This function should accept a string and a substring to compare against. The number of occurrences of the second parameter within the first parameter string are returned. What this means is that we are checking the number of occurrences of the shorter string (second parameter) within the longer string (first parameter). One way to accomplish this is using the `split` function. Here is how to do that:

1. Define the function to accept two parameters. `word` for the input string and `x` for the second string we are searching for
2. Split the string into substrings based on the second input parameter
3. Count the number of instances the substring appeared in the first input string based on the split string
4. Return the result



Write a function named `count_multi_char_x` that takes a string named `word` and a string named `x`. This function should do the same thing as the `count_char_x` function you just wrote - it should return the number of times `x` appears in `word`. However, this time, make sure your function works when `x` is multiple characters long.

For example, `count_multi_char_x("Mississippi", "iss")` should return `2`

▼ **Hint**

Consider using the `split` function. How does the length of `word.split(x)` relate to the number of times `x` was in `word`?

```python
# Write your count_multi_char_x function here:
def count_multi_char_x(word, x):
  word_list = word.split(x)
  new_word = ''.join(word_list)
  first_result = len(word) - len(new_word)
  return int(first_result/len(x))

# Uncomment these function calls to test your function:
print(count_multi_char_x("mississippi", "iss"))
# should print 2
print(count_multi_char_x("apple", "pp"))
# should print 1
```

```
2
1
```

Here is one possible way to get the answer:

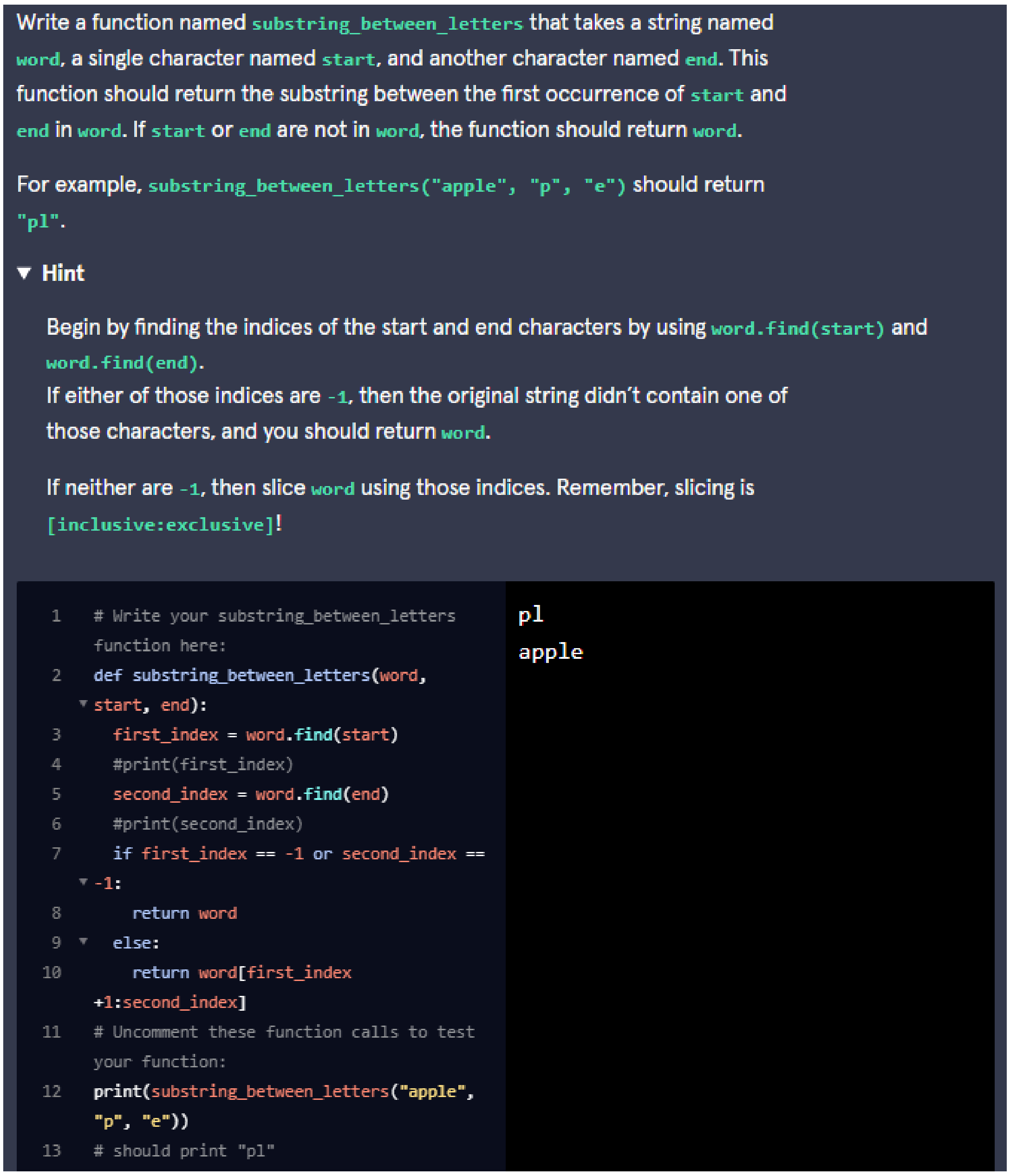
```python
def count_multi_char_x(word, x):
  splits = word.split(x)
  return(len(splits)-1)
```

In our function, we `split` the first input string using the second input string. What this does is cut the first string into an array of smaller substrings containing the parts not equal to our second parameter `x`. For example, when splitting `"mississippi"` with the string `"iss"`, the resulting array will be [“m”, “”, “ippi”]. This includes the characters before `"iss"` was found, the empty space between the two instances of `"iss"` and the characters after the last`"iss"`. Be careful! In order to get the correct result we need to return one less than the total number of split sections — in this example, `"iss"` was in the string twice, resulting in 3 sections. So we should return `3 - 1`.

4. Substring Between

Here is a challenging problem. We need a function that is able to extract a portion of a string between two characters. The function will take three parameters where the first parameter is the string we are going to extract the substring from, the second input is the starting character of our substring and the third character is the ending character of our substring. Here are the steps we can use:

1. Define the function to accept three parameters, one string and two characters
2. `find` the starting index of our substring using the second input parameter
3. `find` the ending index of our substring using the third input parameter
4. If neither of the indices are -1, return the portion of the first input parameter string between the starting and ending indices (not including the starting and ending index)
5. If either of the indices are -1, that means the start or end of the substring didn’t exist in our string. Return the original string in this case.



Write a function named `substring_between_letters` that takes a string named `word`, a single character named `start`, and another character named `end`. This function should return the substring between the first occurrence of `start` and `end` in `word`. If `start` or `end` are not in `word`, the function should return `word`.

For example, `substring_between_letters("apple", "p", "e")` should return `"pl"`.

▼ **Hint**

Begin by finding the indices of the start and end characters by using `word.find(start)` and `word.find(end)`.
If either of those indices are `-1`, then the original string didn't contain one of those characters, and you should return `word`.

If neither are `-1`, then slice `word` using those indices. Remember, slicing is `[inclusive:exclusive]`!

```python
# Write your substring_between_letters function here:
def substring_between_letters(word, start, end):
  first_index = word.find(start)
  #print(first_index)
  second_index = word.find(end)
  #print(second_index)
  if first_index == -1 or second_index == -1:
    return word
  else:
    return word[first_index+1:second_index]
# Uncomment these function calls to test your function:
print(substring_between_letters("apple", "p", "e"))
# should print "pl"
```

```
pl
apple
```

Here is this solution:

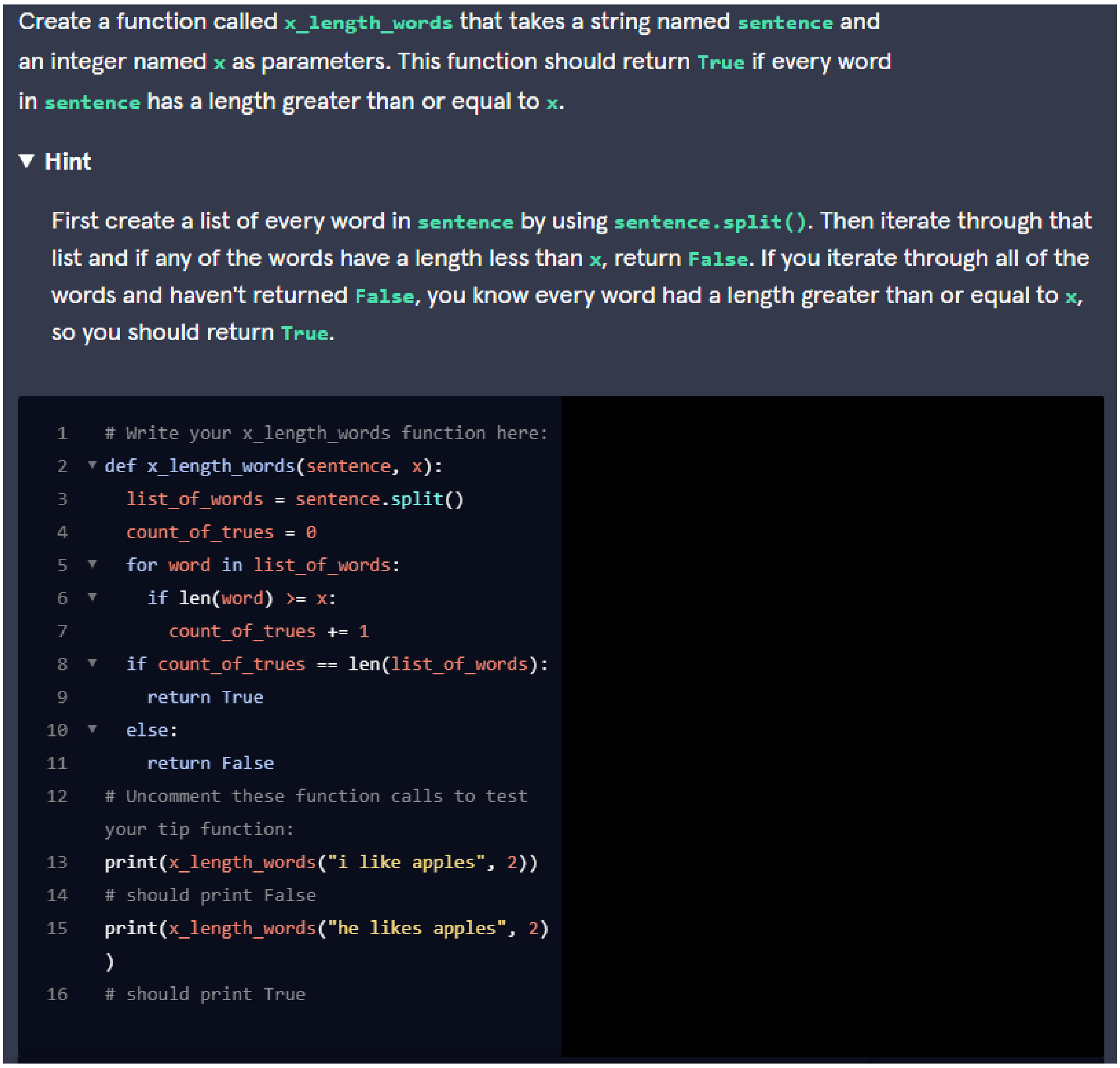
```python
def substring_between_letters(word, start, end):
  start_ind = word.find(start)
  end_ind = word.find(end)
  if start_ind > -1 and end_ind > -1:
    return(word[start_ind+1:end_ind])
  return word
```

In this solution, we use the `find` function to get the starting and ending indices of our substring using our starting and ending characters. After getting those, we check to make sure neither of them are -1. In order to extract the portion of the string within those indices, we use slicing. We provide the starting index plus one in order to not include the starting character. We do not need to provide the end index plus one, since the value on the right of the colon is excluded. This causes our slicing to look like: `word[start_ind+1:end_ind])`.

5. X Length

Let’s use the `split` method in a different way. We need a new function that is able to accept two inputs: one for a sentence and another for a number. The function returns `True` if every single word in the sentence has a length greater than or equal to the number provided. These are the steps:

1. Define the function to accept two parameters, one string, and one number
2. Split up the sentence into an array of words
3. Loop through the words. If the length of any of the words is less than the provided number return `False`
4. If we made it through the loop without returning `False` then return `True`



Create a function called `x_length_words` that takes a string named `sentence` and an integer named `x` as parameters. This function should return `True` if every word in `sentence` has a length greater than or equal to `x`.

▼ Hint

First create a list of every word in `sentence` by using `sentence.split()`. Then iterate through that list and if any of the words have a length less than `x`, return `False`. If you iterate through all of the words and haven't returned `False`, you know every word had a length greater than or equal to `x`, so you should return `True`.

```
# Write your x_length_words function here:
def x_length_words(sentence, x):
  list_of_words = sentence.split()
  count_of_trues = 0
  for word in list_of_words:
    if len(word) >= x:
      count_of_trues += 1
  if count_of_trues == len(list_of_words):
    return True
  else:
    return False
# Uncomment these function calls to test your tip function:
print(x_length_words("i like apples", 2))
# should print False
print(x_length_words("he likes apples", 2))
# should print True
```

Here is what we did:

```
def x_length_words(sentence, x):
  words = sentence.split(" ")
  for word in words:
    if len(word) < x:
      return False
  return True
```

We can use the `split` function with the space character provided in order to get an array of all of the words in the sentence. Next, we use the `in` keyword in order to loop through every element in our array of words. We check the length of each word and compare it against `x` to see if it is shorter. If any of the words in the array are shorter then we immediately return `False` and end the function. If we make through all of the words without returning `False`, we know we should return `True` since all of the word’s lengths were longer than `x`.