Python Code Challenges: Strings (Advanced)

Difficult Python Code Challenges Involving Strings

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

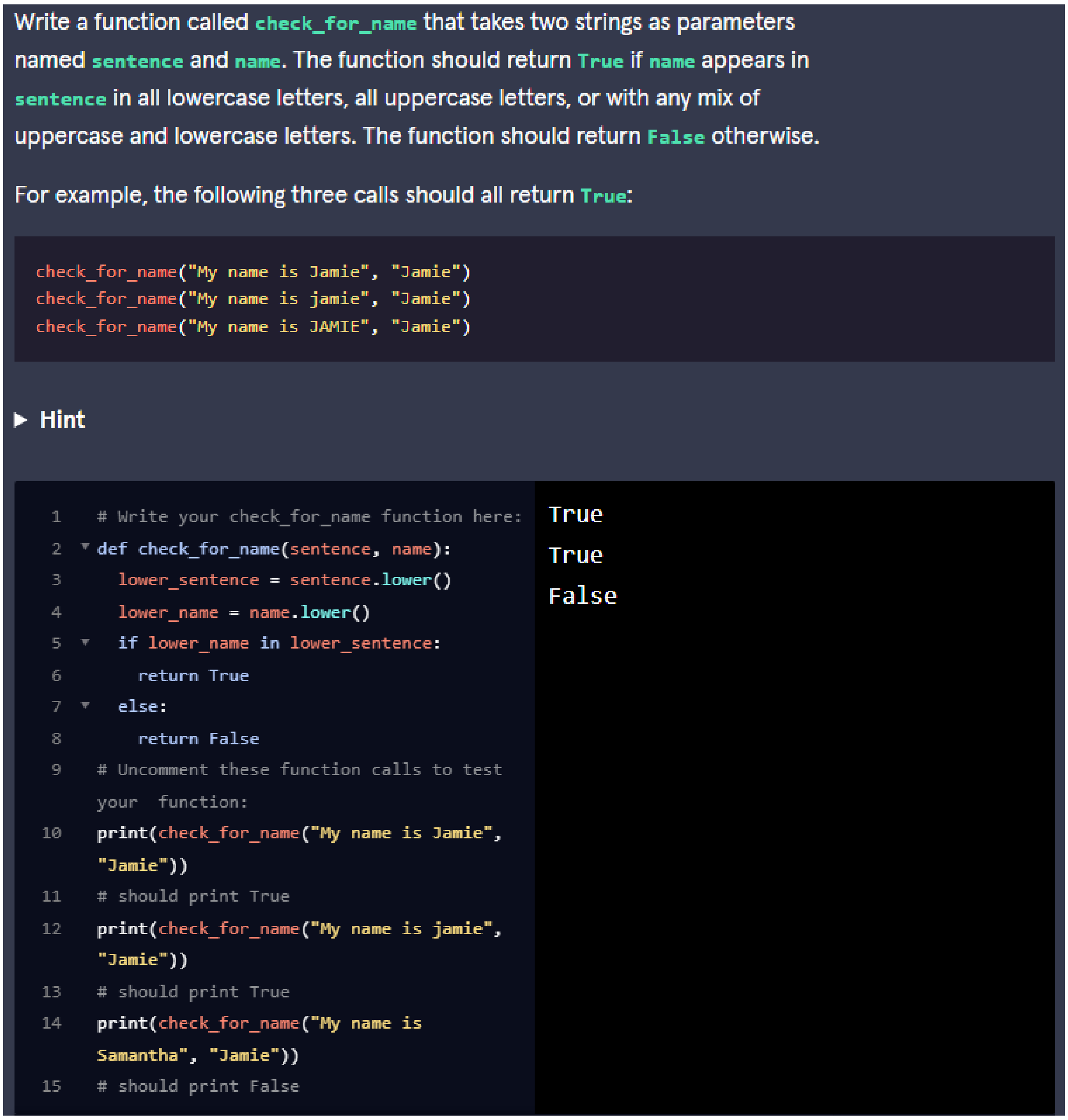
Challenges

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

1. Check Name

You are creating an app that allows users to interact and share their coding project ideas online. The first step is to provide your name in the application and a greeting for other people to see which contains your name. Let's create a function that is able to check if a user's name is located within their greeting. We need a function that accepts two parameters, a string for our sentence and a string for a name. The function should return `True` if the name exists within the string (ignoring any differences in capitalization). Here is what we need to do:

1. Define the function to accept two parameters, one string for the sentence and one string for the name
2. Convert all of the strings to the same case so we don't have to worry about differences in capitalization
3. Check if the name is within the sentence. If so, then return `True`. Otherwise, return `False`


Write a function called `check_for_name` that takes two strings as parameters named `sentence` and `name`. The function should return `True` if `name` appears in `sentence` in all lowercase letters, all uppercase letters, or with any mix of uppercase and lowercase letters. The function should return `False` otherwise.

For example, the following three calls should all return `True`:

```
check_for_name("My name is Jamie", "Jamie")
check_for_name("My name is jamie", "Jamie")
check_for_name("My name is JAMIE", "Jamie")
```

▶ **Hint**

```
# Write your check_for_name function here:
def check_for_name(sentence, name):
  lower_sentence = sentence.lower()
  lower_name = name.lower()
  if lower_name in lower_sentence:
    return True
  else:
    return False
# Uncomment these function calls to test your  function:
print(check_for_name("My name is Jamie", "Jamie"))
# should print True
print(check_for_name("My name is jamie", "Jamie"))
# should print True
print(check_for_name("My name is Samantha", "Jamie"))
# should print False
```

```
True
True
False
```

Here is how we did it:

```
def check_for_name(sentence, name):
  return name.lower() in sentence.lower()
```

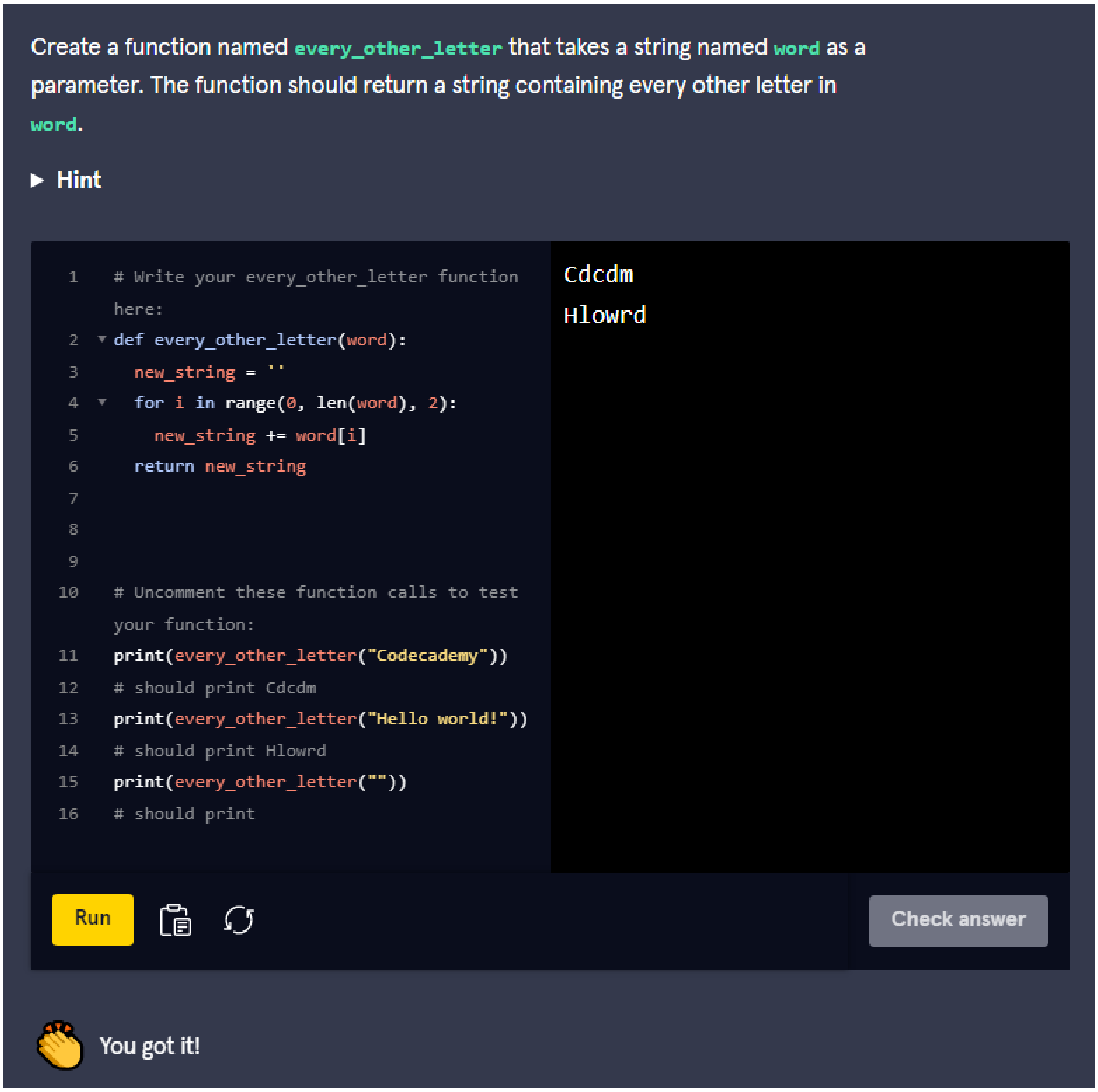
As you can see, this function can be created using one line. The `in` keyword will return `True` if the first provided string is within the second. So in this case, we're checking if `name` is in `sentence`. In order to ignore differences in capitalization, we can use the `.lower()` function which converts all characters to lowercase characters.

2. Every Other Letter

For this next function, we are going to create a function that extract every other letter from a string and returns the resulting string. There are a few different ways you can solve this problem Here are the steps needed for one of the ways:

1. Define the function to accept one parameter for the string

2. Create a new empty string to hold every other letter from the input string
3. Loop through the input string while incrementing by two every time
4. Inside the loop, append the character at the current location to the new string we initialized earlier
5. Return the new string



Create a function named `every_other_letter` that takes a string named `word` as a parameter. The function should return a string containing every other letter in `word`.

```
# Write your every_other_letter function here:
def every_other_letter(word):
  new_string = ''
  for i in range(0, len(word), 2):
    new_string += word[i]
  return new_string



# Uncomment these function calls to test your function:
print(every_other_letter("Codecademy"))
# should print Cdcdm
print(every_other_letter("Hello world!"))
# should print Hlowrd
print(every_other_letter(""))
# should print
```

```
Cdcdm
Hlowrd
```

Here is one way to do it:

```
def every_other_letter(word):
  every_other = ""
  for i in range(0, len(word), 2):
    every_other += word[i]
  return every_other
```

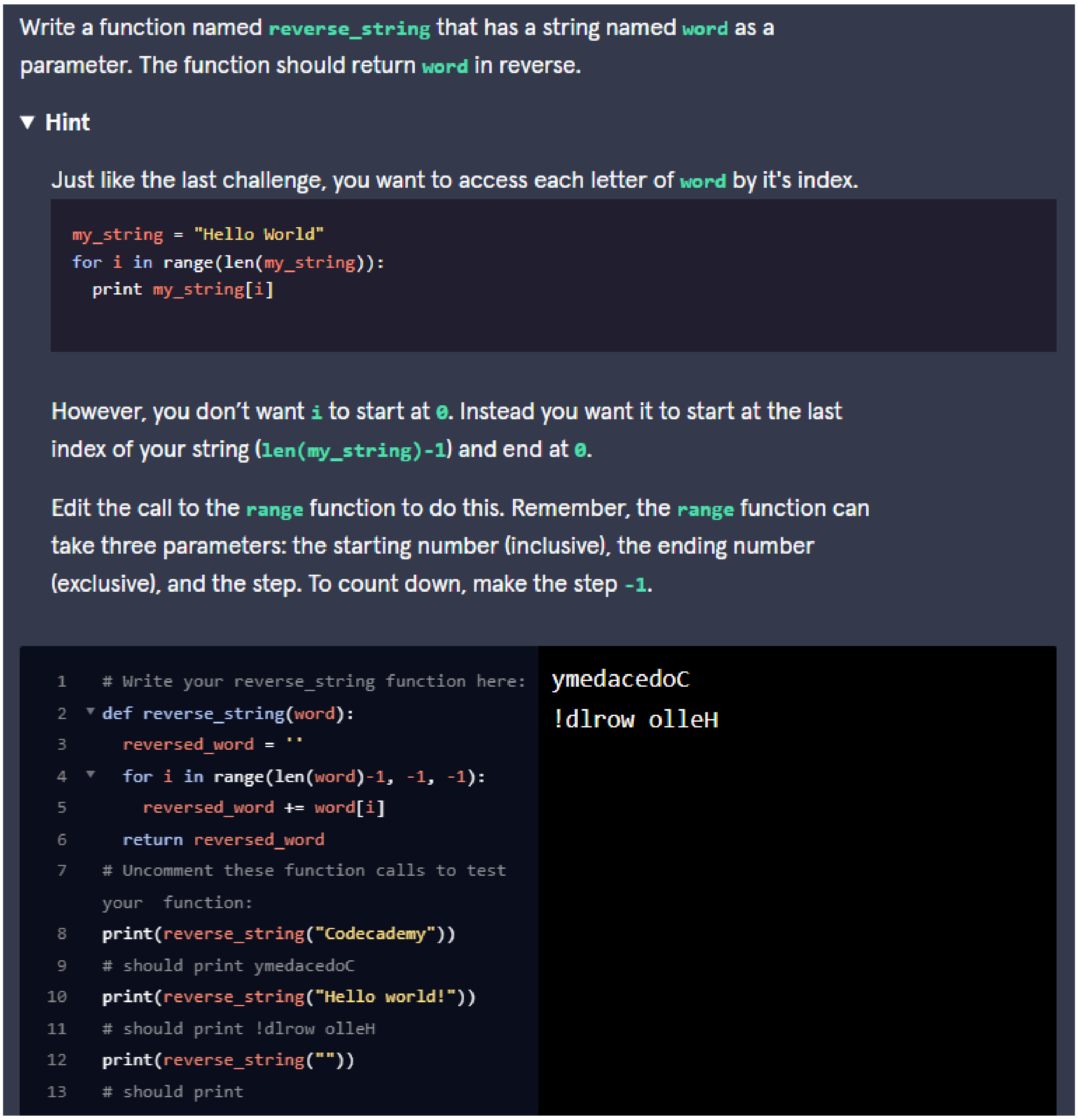
In order to alternate which character is added to the `every_other` string, we use a `range` of indices which starts at index 0 (the beginning of our `word`) and ends at the end of our `word`. The third parameter in the `range` function determines what value to increment by. In this case, we want to increment by 2 in order to get every other letter.

Another way to loop through all indices of our original string, but only add the letters that correspond to an even index. As a challenge, try solving this problem that way!

3. Reverse

This one is similar to the last challenge. Instead of selecting every other letter, we want to reverse the entire string. This can be performed in a similar manner, but we will need to modify the `range` we are using. Here is what we need to do:

1. Define the function to accept one parameter for the string
2. Create a new empty string to hold the reversed string
3. Loop through the input string by starting at the end, and working towards the beginning
4. Inside the loop, append the character at the current location to the new string we initialized earlier
5. Return the result


Write a function named `reverse_string` that has a string named `word` as a parameter. The function should return `word` in reverse.

▼ **Hint**

Just like the last challenge, you want to access each letter of `word` by it's index.

```
my_string = "Hello World"
for i in range(len(my_string)):
  print my_string[i]
```

However, you don't want `i` to start at `0`. Instead you want it to start at the last index of your string (`len(my_string)-1`) and end at `0`.

Edit the call to the `range` function to do this. Remember, the `range` function can take three parameters: the starting number (inclusive), the ending number (exclusive), and the step. To count down, make the step `-1`.

```
# Write your reverse_string function here:
def reverse_string(word):
  reversed_word = ''
  for i in range(len(word)-1, -1, -1):
    reversed_word += word[i]
  return reversed_word
# Uncomment these function calls to test your  function:
print(reverse_string("Codecademy"))
# should print ymedacedoC
print(reverse_string("Hello world!"))
# should print !dlrow olleH
print(reverse_string(""))
# should print
```

```
ymedacedoC
!dlrow olleH
```

Here is this solution:

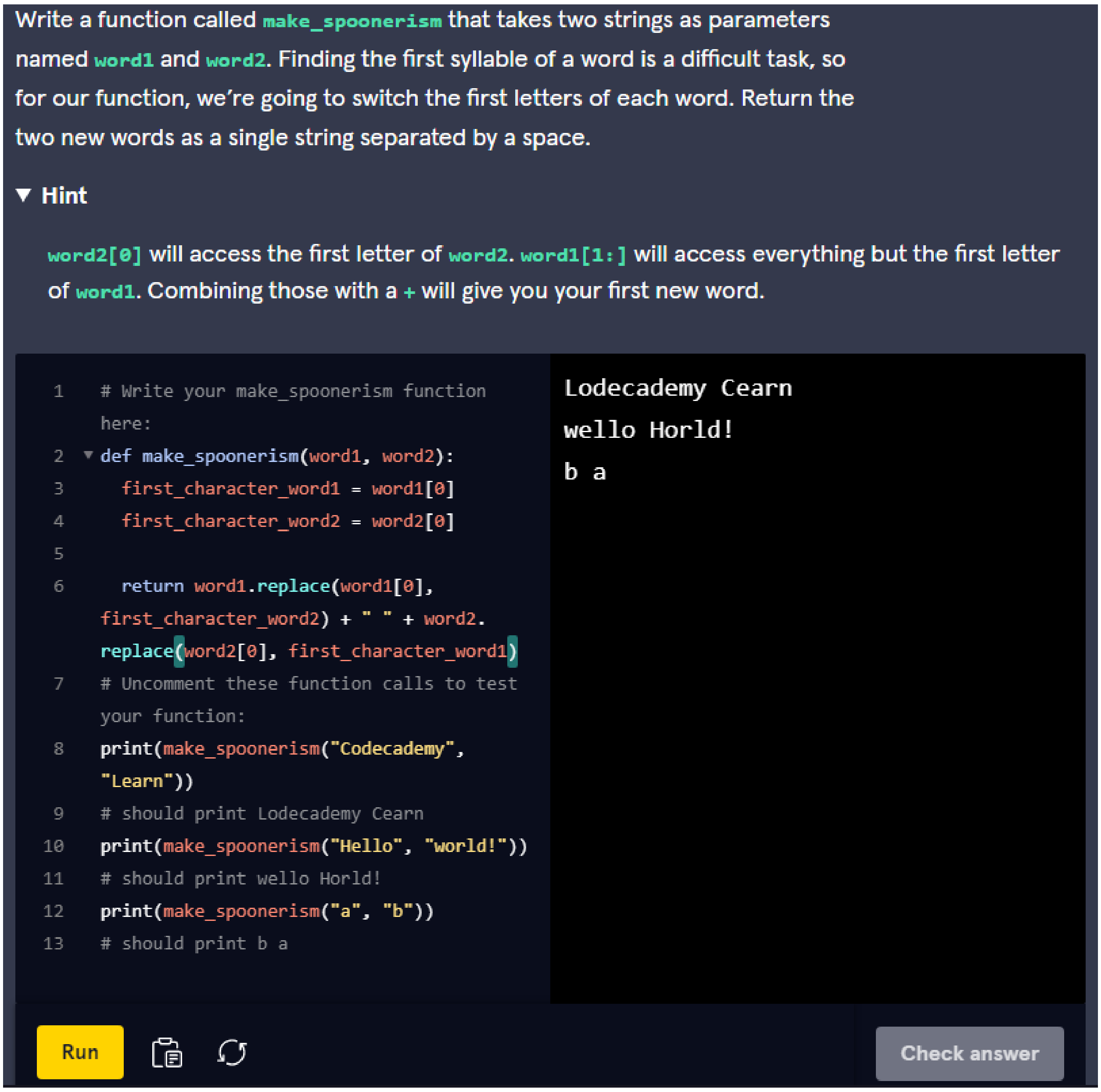
```
def reverse_string(word):
  reverse = ""
  for i in range(len(word)-1, -1, -1):
    reverse += word[i]
  return reverse
```

This is similar to the last solution, but our `range` has been modified in order to start at the last index of the string (length of the string minus one) up to the first index. Since the parameter for the ending index is exclusive we need to provide the index of one more iteration than what we want to stop at. We want to stop at `0`, and since we are incrementing by -1, we will set the ending index to -1. Finally, make sure to add the third parameter of `-1`. This makes us increment by `-1` at each step.

4. Make Spoonerism

A Spoonerism is an error in speech when the first syllables of two words are switched. For example, a Spoonerism is made when someone says “Belly Jeans” instead of “Jelly Beans”. We are going to make a function that is similar, but instead of using the first syllable, we are going to switch the first character of two words. Here is what we need to do:

1. Define the function to accept two parameters for our two words
2. Get the first character of the first word and the first character of the second word
3. Get the remaining characters of the first word and the remaining characters of the second word.
4. Append the first character of the second word to the remaining character of the first word
5. Append a space character
6. Append the first character of the first word to the remaining characters of the second word.
7. Return the result



Write a function called `make_spoonerism` that takes two strings as parameters named `word1` and `word2`. Finding the first syllable of a word is a difficult task, so for our function, we're going to switch the first letters of each word. Return the two new words as a single string separated by a space.

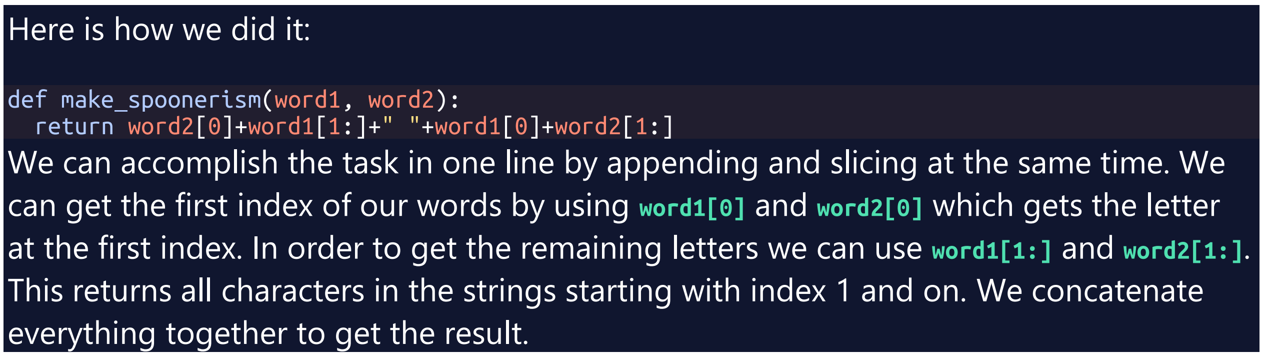
▼ **Hint**

`word2[0]` will access the first letter of `word2`. `word1[1:]` will access everything but the first letter of `word1`. Combining those with a `+` will give you your first new word.

```python
# Write your make_spoonerism function here:
def make_spoonerism(word1, word2):
  first_character_word1 = word1[0]
  first_character_word2 = word2[0]

  return word1.replace(word1[0], first_character_word2) + " " + word2.replace(word2[0], first_character_word1)
# Uncomment these function calls to test your function:
print(make_spoonerism("Codecademy", "Learn"))
# should print Lodecademy Cearn
print(make_spoonerism("Hello", "world!"))
# should print wello Horld!
print(make_spoonerism("a", "b"))
# should print b a
```

```
Lodecademy Cearn
wello Horld!
b a
```



Here is how we did it:

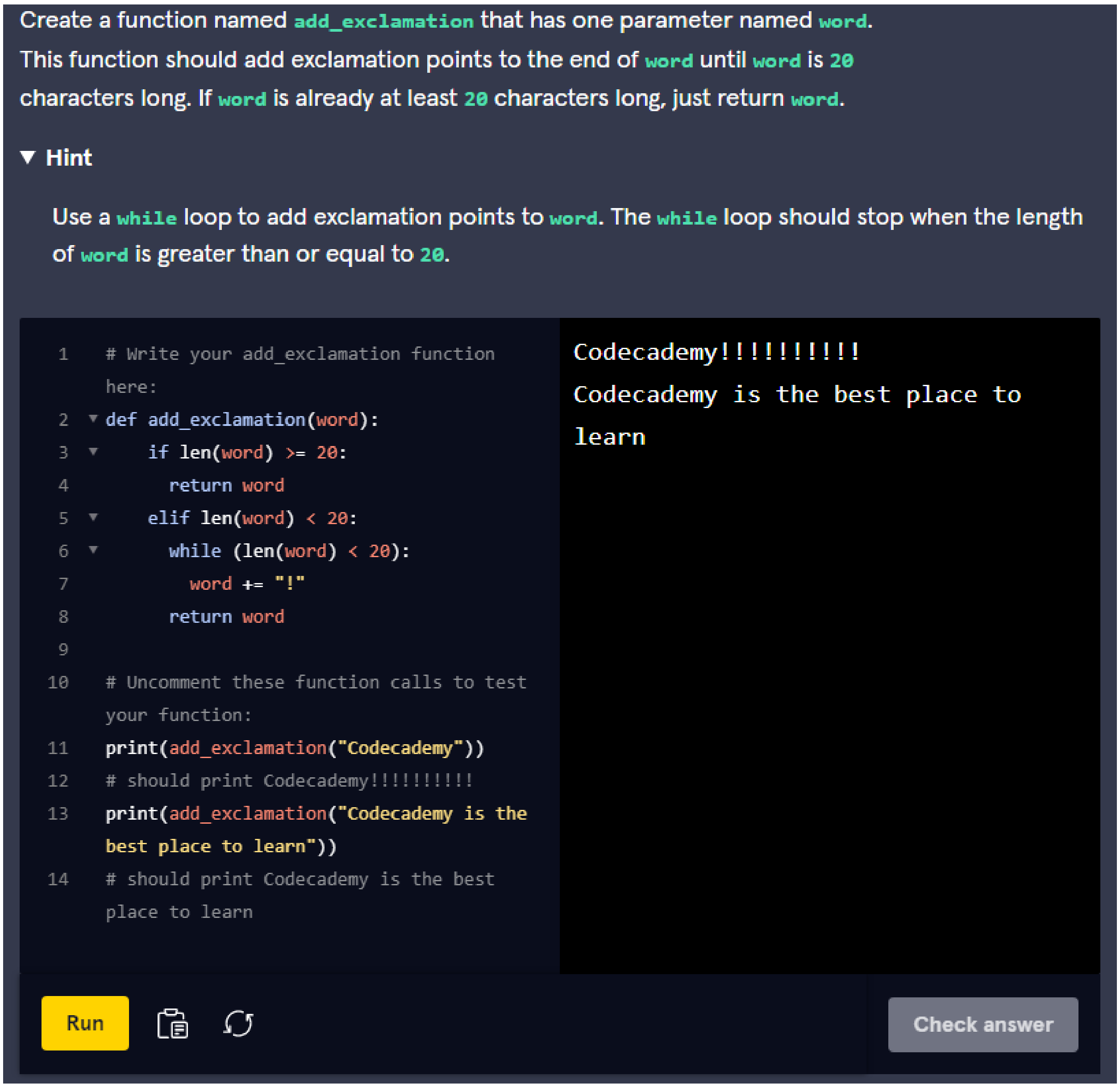
```python
def make_spoonerism(word1, word2):
  return word2[0]+word1[1:]+" "+word1[0]+word2[1:]
```

We can accomplish the task in one line by appending and slicing at the same time. We can get the first index of our words by using `word1[0]` and `word2[0]` which gets the letter at the first index. In order to get the remaining letters we can use `word1[1:]` and `word2[1:]`. This returns all characters in the strings starting with index 1 and on. We concatenate everything together to get the result.

5. Add Exclamation

Let's say we are writing a program that displays a large message on a blimp and we need to fill in any missing space if a short phrase is provided. Our function will accept a string and if the size is less than 20, it will fill in the remaining space with exclamation marks until the size reaches 20. If the provided string already has a length greater than 20, then we will simply return the original string. Here are the steps:

1. Define the function to accept one parameter for our string
2. Loop while the length of our input string is less than 20
3. Inside the loop, append an exclamation mark
4. Once done, return the result



Create a function named `add_exclamation` that has one parameter named `word`.
This function should add exclamation points to the end of `word` until `word` is `20` characters long. If `word` is already at least `20` characters long, just return `word`.

▼ Hint

Use a `while` loop to add exclamation points to `word`. The `while` loop should stop when the length of `word` is greater than or equal to `20`.

```python
# Write your add_exclamation function here:
def add_exclamation(word):
  if len(word) >= 20:
    return word
  elif len(word) < 20:
    while (len(word) < 20):
      word += "!"
    return word

# Uncomment these function calls to test your function:
print(add_exclamation("Codecademy"))
# should print Codecademy!!!!!!!!!!
print(add_exclamation("Codecademy is the best place to learn"))
# should print Codecademy is the best place to learn
```

```
Codecademy!!!!!!!!!!
Codecademy is the best place to learn
```

Here is this solution:

```
def add_exclamation(word):
  while(len(word) < 20):
    word += "!"
  return word
```

This function shows how we can continuously append to our string based on some condition. In this case, we keep testing the length of the string to see if we should keep going. Once the length has reached 20, either by adding exclamation marks or by already being long, we return the result.