Python Code Challenges: Lists

Python Code Challenges involving Lists

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

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 returns the sum of the first and last elements of a given list might look like this:

```
def first_plus_last(lst):
  return lst[0] + lst[-1]
```

And this would produce output like:

```
>>> first_plus_last([1, 2, 3, 4])
5
>>> first_plus_last([8, 2, 5, -8])
0
>>> first_plus_last([-10, 2, 3, -4])
-14
```

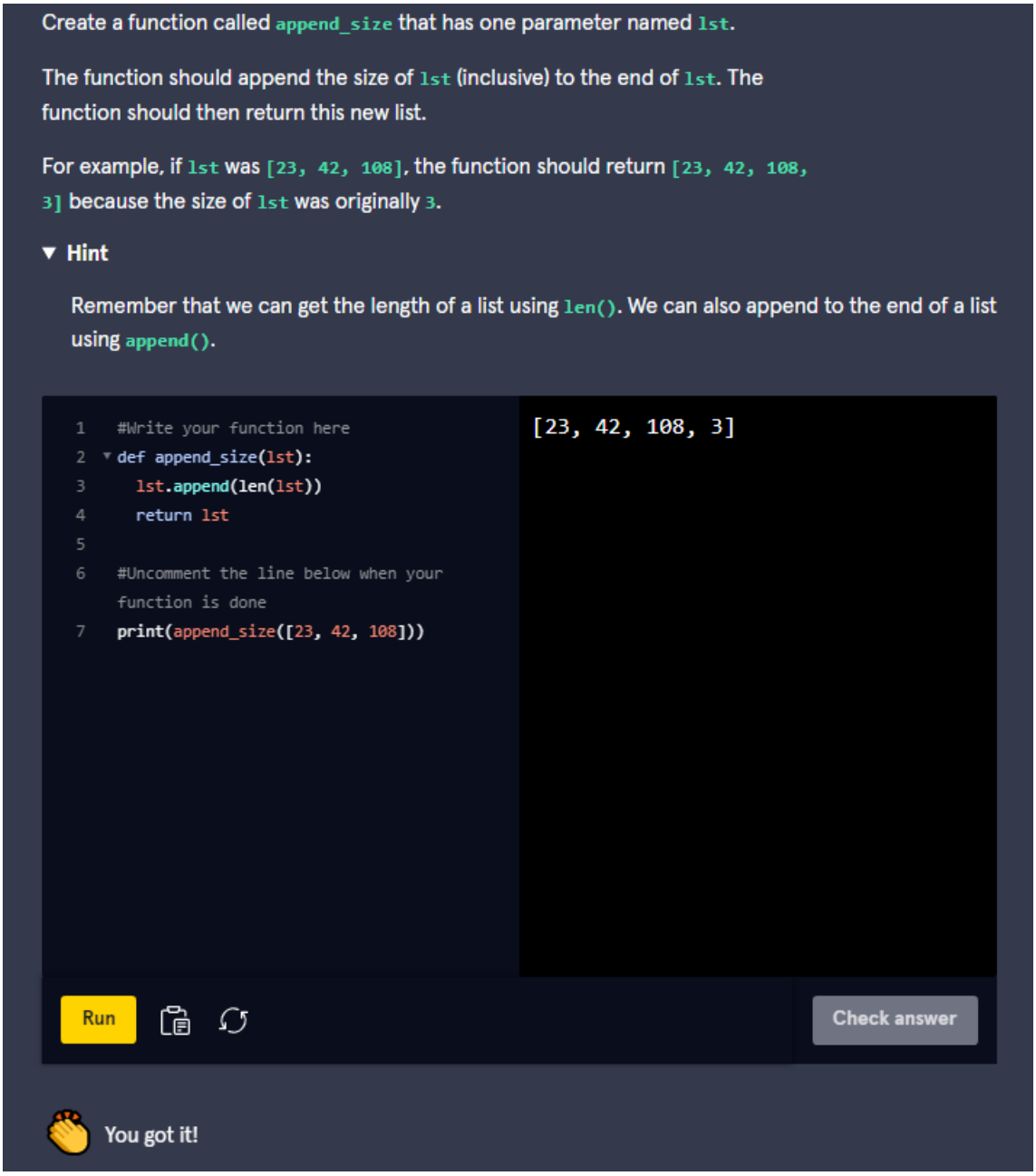
Challenges

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

1. Append Size

For the first code challenge, we are going to calculate the length of a list and then append the value to the end of the list. Here is what we need to do:

1. Define the function to accept one parameter for our list
2. Get the length of the list
3. Append the length of the list to the end of the list
4. Return the modified list



Here is this solution:

```
def append_size(lst):
  lst.append(len(lst))
  return lst
```

We can get the length and append it at the same time by nesting the function calls as shown in the solution. Afterward, we return the modified list.

2. Append Sum

Let's create a function that calculates the sum of the last two elements of a list and appends it to the end. After doing so, it will repeat this process two more times and return the resulting list. You can choose to use a loop or manually use three lines. Here are the steps we need:

1. Define the function to accept one parameter for our list of numbers
2. Add the last and second to last elements from our list together
3. Append the calculated value to the end of our list.
4. Repeat steps 2 and 3 two more times
5. Return the modified list

Write a function named `append_sum` that has one parameter — a list named named `lst`.

The function should add the last two elements of `lst` together and append the result to `lst`. It should do this process three times and then return `lst`.

For example, if `lst` started as `[1, 1, 2]`, the final result should be `[1, 1, 2, 3, 5, 8]`.

▼ Hint

In order to get the last element of a list we can use `lst[-1]`. The negative index starts from the end of the list and moves towards the front. Because of this, to get the second to last number we can use `lst[-2]`. Additionally, to append items to a list we can use the `append()` function.

```
#Write your function here
def append_sum(lst):
  for temp in range(3):
    lst.append(lst[-1] + lst[-2])
  return lst


#Uncomment the line below when your function is done
print(append_sum([1, 1, 2]))
```

```
[1, 1, 2, 3, 5, 8]
```

Run

Check answer

This is how we solved it:

```
def append_sum(lst):
  lst.append(lst[-1] + lst[-2])
  lst.append(lst[-1] + lst[-2])
  lst.append(lst[-1] + lst[-2])
  return lst
```

In our solution, we add the numbers and append the result in one line. We add the last and second to last elements within the `.append()` function and we repeat this line two more times. Remember that when we use negative indices, it starts from the end of the list and goes towards the beginning of the list. You could also use a loop to solve this instead of repeating the lines.

3. Larger List

Let's say we are working with two conveyor belts that contain items represented by a numerical ID. If one conveyor belt contains more items than the other, then we need to return the ID of the last item on that belt. In the case where they have the same number of items, return the last item from the first conveyor belt. In our code, we can represent the belts using lists. Here are the steps:

1. Define the function to accept two parameters for our two lists of numbers
2. Check if the length of the first list is greater than or equal to the length of the second list
3. If true, then return the last element from the first list. Otherwise, return the last element from the second list

Write a function named `larger_list` that has two parameters named `lst1` and `lst2`.

The function should return the last element of the list that contains more elements. If both lists are the same size, then return the last element of `lst1`.

▼ Hint

Remember that we can use `len(lst1)` to get the length of a list called `lst1`. Also, to get the last element of `lst1` we can use `lst1[-1]`.

```
#Write your function here
def larger_list(list1, list2):
  if len(list1) >= len(list2):
    return list1[-1]
  else:
    return list2[-1]

#Uncomment the line below when your function is done
print(larger_list([4, 10, 2, 5], [-10, 2, 5, 10]))
```

```
5
```

Run

Check answer

Here is how we did it:

```
def larger_list(lst1, lst2):
  if len(lst1) >= len(lst2):
    return lst1[-1]
```

```
  else:
    return lst2[-1]
```

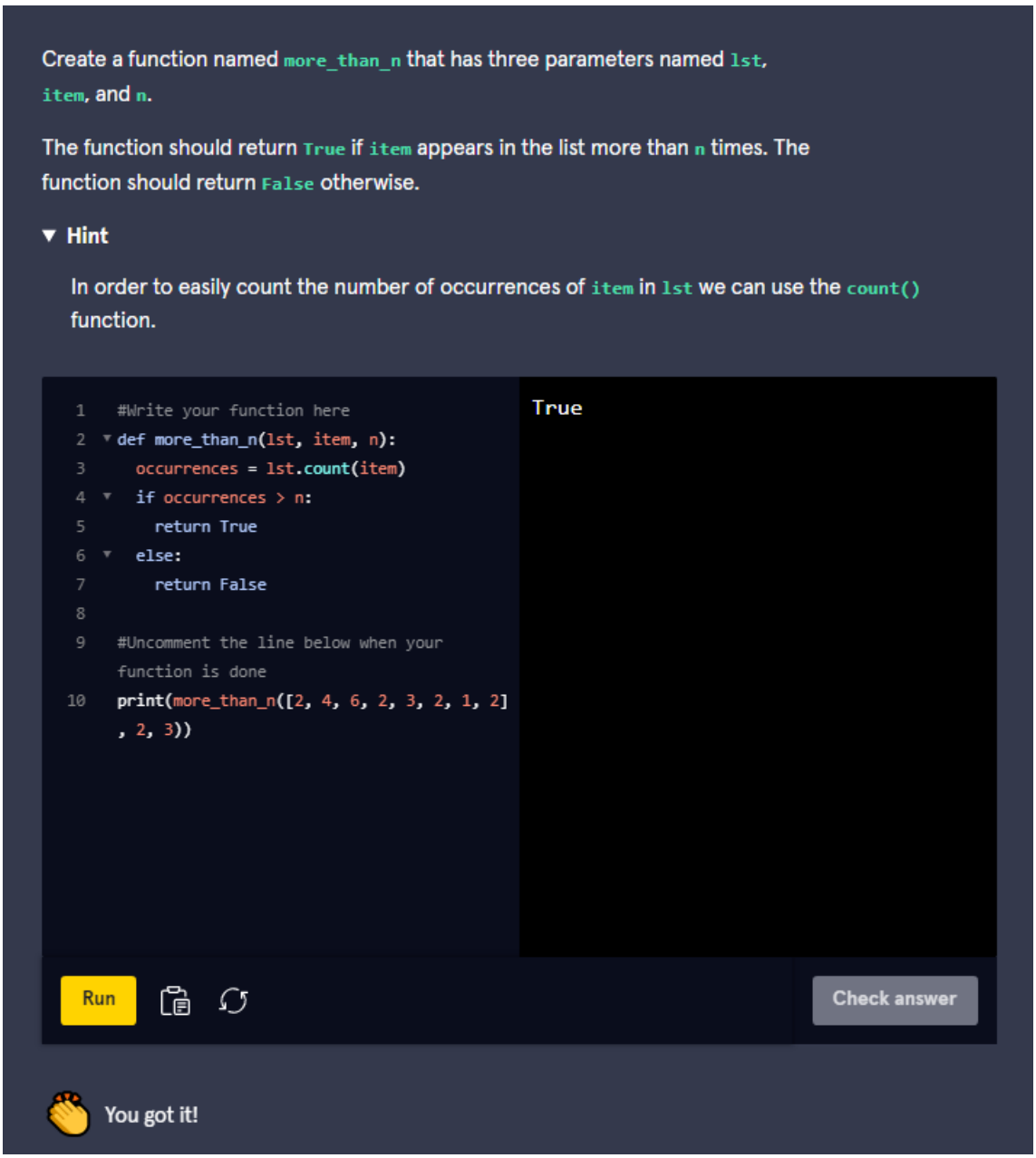
We start by comparing the lengths of each of the lists using the `len()` function. This determines whether to return the last element of the first list or the second list. Notice that we use `>=`. This way, we know what to do if the lists have an equal length.

In order to get the last element, we get the element at the -1 index. The negative index starts at the end of the list and works towards the start of the list.

4. More Than N

Our factory produces a variety of different flavored snacks and we want to check the number of instances of a certain type. We have a conveyor belt full of different types of snacks represented by different numbers. Our function will accept a list of numbers (representing the type of snack), a number for the second parameter (the type of snack we are looking for), and another number as the third parameter (the maximum number of that type of snack on the conveyor belt). The function will return `True` if the snack we are searching for appears more times than we provided as our third parameter. These are the steps we need:

1. Define the function to accept three parameters, a list of numbers, a number to look for, and a number for the number of instances
2. Count the number of occurrences of `item` (the second parameter) in `lst` (the first parameter)
3. If the number of occurrences is greater than `n` (the third parameter), return `True`. Otherwise, return `False`



Here is one way to do it:

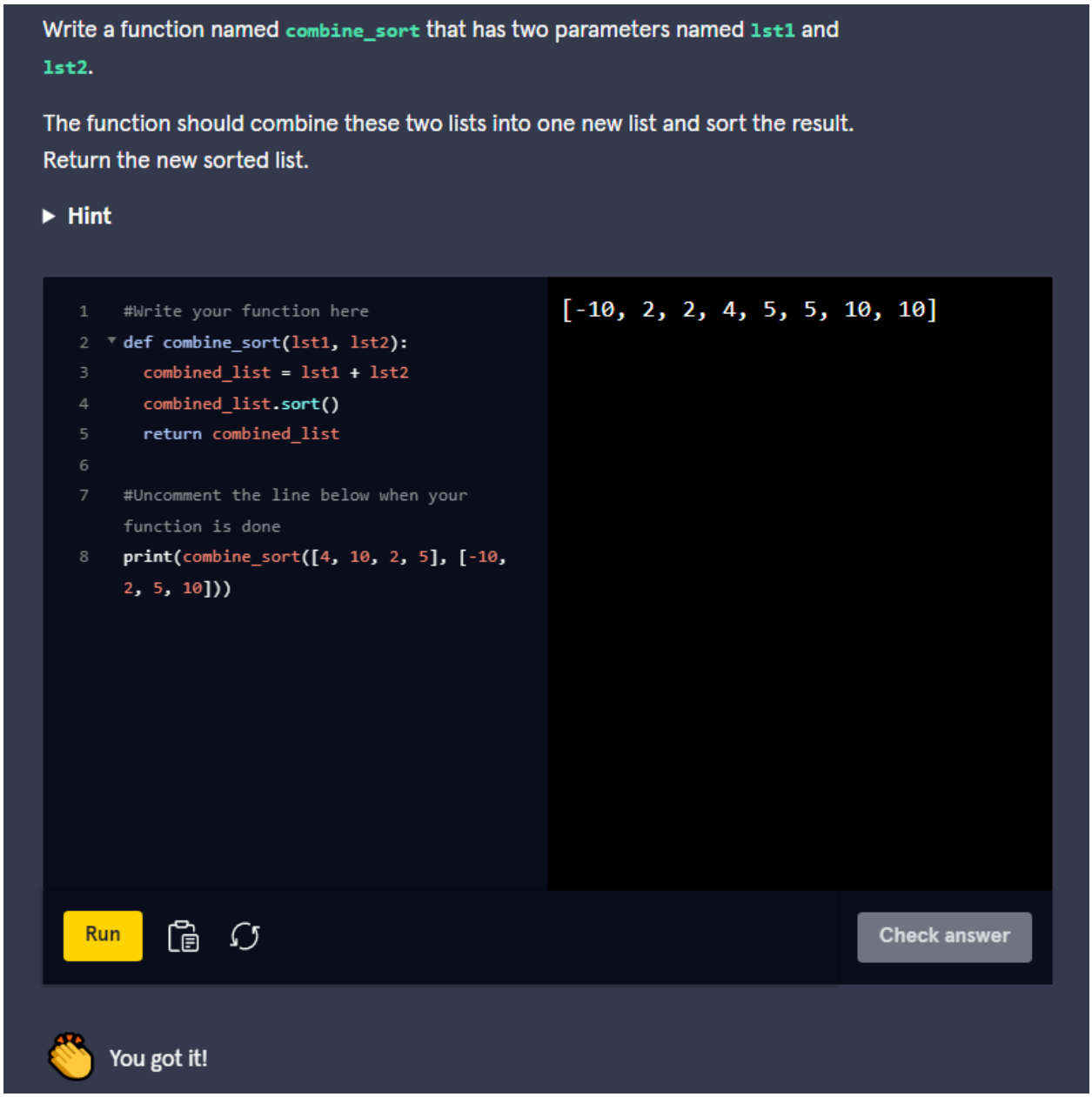
```
def more_than_n(lst, item, n):
  if lst.count(item) > n:
    return True
  else:
    return False
```

We use the `count()` function to count the number of times `item` appears in `lst`. You could also do this manually by looping through `lst` and incrementing a variable every time you see `item`. We then compare the result to `n`.

5. Combine Sort

Finally, let's create a function that combines two different lists together and then sorts them. To do this we can combine the lists with an operation and then sort using a function call. Here are the steps we need to use:

1. Define the function to accept two parameters, one for each list.
2. Combine the two lists together
3. Sort the result
4. Return the sorted and combined list



Here is how we did it:

```
def combine_sort(lst1, lst2):
  unsorted = lst1 + lst2
  sortedList = sorted(unsorted)
  return sortedList
```

We start by combining the two lists together using `+` in order to get a new list. Next, in order to sort them, we use the `sorted()` function which returns a new sorted version of the list.