Python Code Challenges: Lists (Advanced)

Difficult 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. Every Three Numbers

Let's start our challenging problems with a function that creates a list of numbers up to 100 in increments of 3 starting from a number that is passed to the function as an input parameter. Here is what we need to do:

1. Define the function to accept one parameter for our starting number
2. Calculate the numbers between the starting number and 100 while incrementing by 3
3. Store the numbers in a list
4. Return the list

Create a function called `every_three_nums` that has one parameter named `start`.

The function should return a list of every third number between `start` and `100` (inclusive). For example, `every_three_nums(91)` should return the list `[91, 94, 97, 100]`. If `start` is greater than `100`, the function should return an empty list.

▼ **Hint**

We can generate the numbers in a certain range by a certain increment using the `range()` function. In order to convert the range sequence into a list, we can pass it into the `list()` function.

```
1   #Write your function here
2   def every_three_nums(start):
3     result = [start]
4     if start > 100:
5       return []
6     else:
7       while result[-1] != 100:
8         start += 3
9         result.append(start)
10    return result
11
12  #Uncomment the line below when your
    function is done
13  print(every_three_nums(91))
```

```
[91, 94, 97, 100]
```

Run

Check answer

You got it!

Here is what we did:

```
def every_three_nums(start):
  return list(range(start, 101, 3))
```

We can write the body of this function in one line by nesting the `range()` function inside of the `list()` function. The range function accepts the starting number, the ending number (exclusive), and the amount to increment by.

2. Remove Middle

Our next function will remove all elements from a list with an index within a certain range. The function will accept a list, a starting index, and an ending index. All elements with an index between the starting and ending index should be removed from the list. Here are the steps:

1. Define the function to accept three parameters: the list, the starting index, and the ending index
2. Get all elements before the starting index
3. Get all elements after the ending index
4. Combine the two partial lists into the result
5. Return the result

Create a function named `remove_middle` which has three parameters named `lst`, `start`, and `end`.

The function should return a list where all elements in `lst` with an index between `start` and `end` (inclusive) have been removed.

For example, the following code should return `[4, 23, 42]` because elements at indices `1`, `2`, and `3` have been removed:

```
remove_middle([4, 8 , 15, 16, 23, 42], 1, 3)
```

▼ Hint

To make this problem easier, we can use slicing. For example, if we wanted all elements up to a certain index we can use `lst[:index]` and to get all elements after a certain index we can use `lst[index+1:]`.

```
#Write your function here
def remove_middle(lst, start, end):
  result = []
  forbidden_indexes = list(range(start, end + 1, 1))
  for index in forbidden_indexes:
    result.append(lst[index])
  for number in result:
    lst.remove(number)
  return lst


#Uncomment the line below when your function is done
print(remove_middle([4, 8, 15, 16, 23, 42], 1, 3))
```

```
[4, 23, 42]
```

Here is what we did:

```
def remove_middle(lst, start, end):
  return lst[:start] + lst[end+1:]
```

This can be solved using one line of code if you combine and slice the lists at the same time. Slicing allows us to get the segments of the list before and after the index range and the operation `+` allows us to combine them together.

3. More Frequent Item

Let's go back to our factory example. We have a conveyor belt of items where each item is represented by a different number. We want to know, out of two items, which one shows up more on our belt. To solve this, we can use a function with three parameters.

One parameter for the list of items, another for the first item we are comparing, and another for the second item. Here are the steps:

1. Define the function to accept three parameters: the list, the first item, and the second item
2. Count the number of times `item1` shows up in our list
3. Count the number of times `item2` shows up in our list
4. Return the item that appears more frequently in `lst` — if both items show up the same number of times, return `item1`

Create a function named `more_frequent_item` that has three parameters named `lst`, `item1`, and `item2`.

Return either `item1` or `item2` depending on which item appears more often in `lst`.

If the two items appear the same number of times, return `item1`.

▼ Hint

Remember that we can easily count the number of occurrences of an item in our list using `lst.count(item1)`.

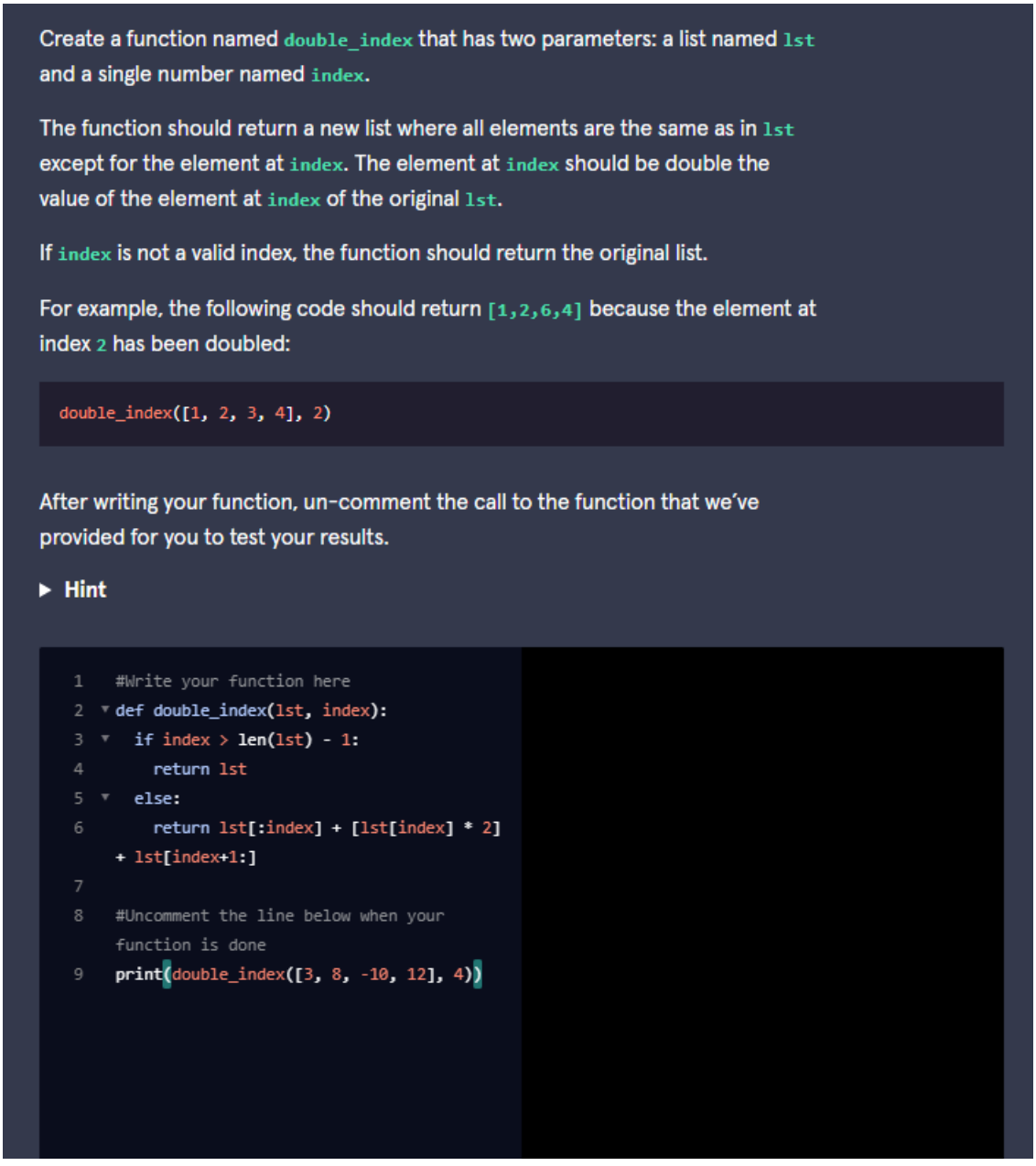
```
#Write your function here
def more_frequent_item(lst, item1, item2):
  if lst.count(item1) > lst.count(item2):
    return item1
  elif lst.count(item2) > lst.count(item1):
    return item2
  else:
    return item1
#Uncomment the line below when your function is done
print(more_frequent_item([2, 3, 3, 2, 3, 2, 3, 2, 3], 2, 3))
```

```
3
```

4. Double Index

Our next function will double a value at a given position. We will provide a list and an index to double. This will create a new list by replacing the value at the index provided with double the original value. If the index is invalid then we should return the original list. Here is what we need to do:

1. Define the function to accept two parameters, one for the list and another for the index of the value we are going to double
2. Test if the index is invalid. If it's invalid then return the original list
3. If the index is valid then get all values up to the index and store it as a new list
4. Append the value at the index times 2 to the new list
5. Add the rest of the list from the index onto the new list
6. Return the new list



Create a function named `double_index` that has two parameters: a list named `lst` and a single number named `index`.

The function should return a new list where all elements are the same as in `lst` except for the element at `index`. The element at `index` should be double the value of the element at `index` of the original `lst`.

If `index` is not a valid index, the function should return the original list.

For example, the following code should return `[1,2,6,4]` because the element at index `2` has been doubled:

```
double_index([1, 2, 3, 4], 2)
```

After writing your function, un-comment the call to the function that we've provided for you to test your results.

▶ Hint

```
#Write your function here
def double_index(lst, index):
    if index > len(lst) - 1:
        return lst
    else:
        return lst[:index] + [lst[index] * 2] + lst[index+1:]

#Uncomment the line below when your function is done
print(double_index([3, 8, -10, 12], 4))
```

Here is one way to do it:

```
def double_index(lst, index):
  # Checks to see if index is too big
  if index >= len(lst):
    return lst
  else:
    # Gets the original list up to index
    new_lst = lst[0:index]
 # Adds double the value at index to the new list
  new_lst.append(lst[index]*2)
  #  Adds the rest of the original list
  new_lst = new_lst + lst[index+1:]
  return new_lst
```

Note that this solution is careful not to modify the original input list. If we were to simply use `lst[index] = lst[index] * 2` then the list that was passed into the function would be modified outside of the function as well. Creating a new list and writing the values to it prevents this from happening. We use slicing to extract the values before and after the index and we append the modified value at the index to our new list.

5. Middle Item

For the final code challenge, we are going to create a function that finds the middle item from a list of values. This will be different depending on whether there are an odd or even number of values. In the case of an odd number of elements, we want this function to return the exact middle value. If there is an even number of elements, it returns the average of the middle two elements. Here is what we need to do:

1. Define the function to accept one parameter for our list of numbers
2. Determine if the length of the list is even or odd
3. If the length is even, then return the average of the middle two numbers
4. If the length is odd, then return the middle number

Create a function called `middle_element` that has one parameter named `lst`.

If there are an odd number of elements in `lst`, the function should return the middle element. If there are an even number of elements, the function should return the average of the middle two elements.

▶ Hint

```
#Write your function here
def middle_element(lst):
  longi = len(lst)
  odd_index = int(longi/2 - 1/2)
  even_answer = (lst[int(longi/2 - 1)] + lst[int(longi/2)]) / 2
  if longi % 2 != 0:
    return lst[odd_index]
  if longi % 2 == 0:
    return even_answer



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

```
-7.0
```

Run

Check answer

Here is how we solved it:

```
def middle_element(lst):
  if len(lst) % 2 == 0:
    sum = lst[int(len(lst)/2)] + lst[int(len(lst)/2) - 1]
    return sum / 2
  else:
    return lst[int(len(lst)/2)]
```

We used modulus to determine if the list had an even or odd number of elements. After determining this, for an odd number of elements, we calculate the middle index and return the middle element from the list. For an even number of elements, we calculate the index of the element close to the middle and the other element close to the middle (by subtracting 1 from the middle calculation). We get the values at those indices and calculate the average.

Note that the math to find the middle index is a bit tricky. In some cases, when we divide by `2`, we would get a double. For example, if our list had `3` items in it, then `3/2` would give us `1.5`. The middle index should be `1`, so in order to go from `1.5` to `1`, we cast `1.5` as an `int`. In total, this is `int(len(lst)/2)`.