**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

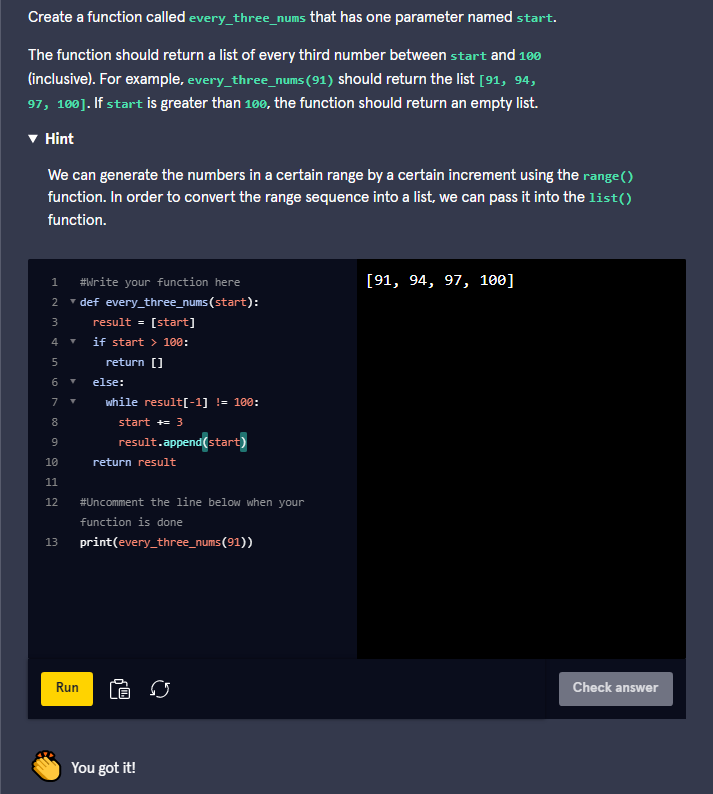
**Challenges**

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



Here is what we did:

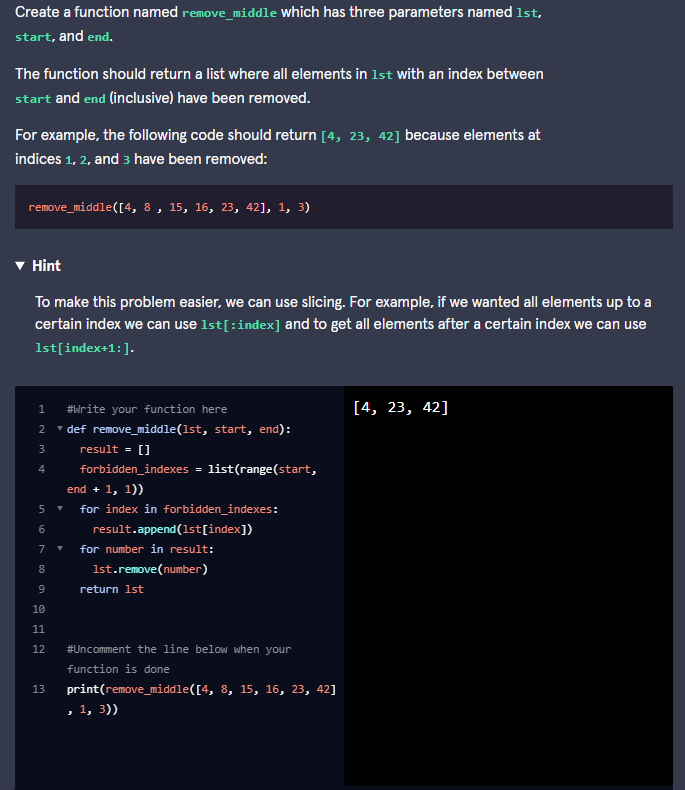
def every\_three\_nums(start):  
  return list(range(start, 101, 3))

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



Here is what we did:

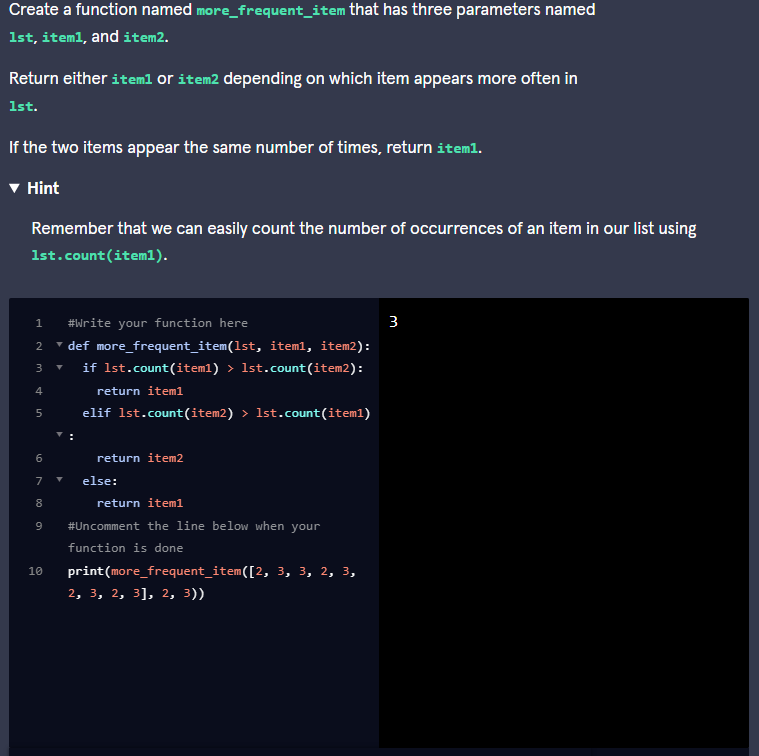
def remove\_middle(lst, start, end):  
  return lst[:start] + lst[end+1:]

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**



### 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



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

