Python Code Challenges: Loops (Advanced)

Difficult Python Code Challenges involving Loops

This lesson will help you review Python loops by providing some challenge exercises involving loops.

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 and Loop 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 prints all odd numbers in a list would look like this:

```
def odd_nums(lst):
  for item in lst:
    if item % 2 == 1:
      print(item)
```

And this would produce output like:

```
>>> odd_nums([4, 7, 9, 10, 13])
7
9
13
```

Challenges

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

1. Larger Sum

We are going to start our advanced challenge problems by calculating which list of two inputs has the larger sum. We will iterate through each of the list and calculate the sums, afterwards we will compare the two and return which one has a greater sum. Here are the steps we need:

1. Define the function to accept two input parameters: `lst1` and `lst2`
2. Create two variables to record the two sums
3. Loop through the first list and add up all of the numbers
4. Loop through the second list and add up all of the numbers
5. Compare the first and second sum and return the list with the greater sum

Create a function named `larger_sum()` that takes two lists of numbers as parameters named `lst1` and `lst2`.

The function should return the list whose elements sum to the greater number. If the sum of the elements of each list are equal, return `lst1`.

▼ Hint

Create variables named `sum1` and `sum2` and set them to be `0`. Loop through each list separately and add to the appropriate variable. After looping through each list, compare the two sums in an if statement and return the correct list.

```
#Write your function here
def larger_sum(lst1, lst2):
  sum1 = 0
  sum2 = 0
  maximus = 0
  for num in lst1:
    sum1 += num
  for num in lst2:
    sum2 += num
  maximus = max(sum1, sum2)
  if maximus == sum1:
    return lst1
  else:
    return lst2


#Uncomment the line below when your function is done
print(larger_sum([1, 9, 5], [2, 3, 7]))
```

```
[1, 9, 5]
```

Run

Check answer

Here's one way to do it:

```
def larger_sum(lst1, lst2):
  sum1 = 0
  sum2 = 0
  for number in lst1:
    sum1 += number
  for number in lst2:
    sum2 += number
  if sum1 >= sum2:
    return lst1
  else:
    return lst2
```

In this solution, we manually iterate through each element in each list and calculate our sums. We then return the list with the greater sum and break the tie by returning `lst1`. You can also try solving this problem using the `sum()` function in python. The body of this function could also be condensed into one line of code if you want an additional challenge!

2. Over 9000

We are constructing a device that is able to measure the power level of our coding abilities and according to the device, it will be impossible for our power levels to be over 9000. Because of this, as we iterate through a list of power values we will count up each of the numbers until our sum reaches a value greater than 9000. Once this happens, we should stop adding the numbers and return the value where we stopped. In order to do this, we will need the following steps:

1. Define the function to accept a list of numbers
2. Create a variable to keep track of our sum
3. Iterate through every element in our list of numbers
4. Within the loop, add the current number we are looking at to our sum
5. Still within the loop, check if the sum is greater than 9000. If it is, end the loop
6. Return the value of the sum when we ended our loop

Create a function named `over_nine_thousand()` that takes a list of numbers named `lst` as a parameter.

The function should sum the elements of the list until the sum is greater than `9000`. When this happens, the function should return the sum. If the sum of all of the elements is never greater than `9000`, the function should return total sum of all the elements. If the list is empty, the function should return `0`.

For example, if `lst` was `[8000, 900, 120, 5000]`, then the function should return `9020`.

▶ **Hint**

```
#Write your function here
def over_nine_thousand(lst):
  elements_sum = 0
  answer = 0
  if len(lst) == 0:
    return 0
  for num in lst:
    elements_sum += num
    if elements_sum > 0 and elements_sum <= 9000:
      answer = elements_sum
    if elements_sum > 9000:
      break
  return elements_sum



#Uncomment the line below when your function is done
```

```
9020
```

Here is this solution:

```
def over_nine_thousand(lst):
  sum = 0
  for number in lst:
    sum += number
    if (sum > 9000):
      break
  return sum
```

Our solution follows a similar pattern to some of the other code challenges, except that we have a condition where we end early. In the case where we reach a sum greater than 9000, we can use the `break` keyword to exit our loop. This will continue to execute the code outside of our loop, which in this case, returns the sum that we found.

3. Max Num

Here is a more traditional coding problem for you. This function will be used to find the maximum number in a list of numbers. This can be accomplished using

the `max()` function in python, but as a challenge, we are going to manually implement this function. Here is what we need to do:

1. Define the function to accept a list of numbers called `nums`
2. Set our default maximum value to be the first element in the list
3. Loop through every number in the list of numbers
4. Within the loop, if we find a number greater than our starting maximum, then replace the maximum with what we found.
5. Return the maximum number

Create a function named `max_num()` that takes a list of numbers named `nums` as a parameter.

The function should return the largest number in `nums`

▼ **Hint**

Create a variable called `maximum` to track the max number, and have it start as the first element in the list. Loop through all of the numbers in the list, and if a number is ever greater than the current max number, the max number should be re-set to that number.

```
#Write your function here
def max_num(nums):
  maximus = nums[0]
  for num in nums:
    if num > maximus:
      maximus = num
  return maximus


#Uncomment the line below when your function is done
print(max_num([50, -10, 0, 75, 20]))
```

```
75
```

Run

Check answer

Here is one way to solve this:

```
def max_num(nums):
  maximum = nums[0]
  for number in nums:
    if number > maximum:
```

```
        maximum = number
    return maximum
```

There are a few different ways to accomplish this task, but the way we did it was to check every element in the list and see if there is one bigger than what we currently think is the biggest. If there is a bigger one, then replace it. We keep replacing the number until the largest number has been found.

4. Same Values

In this challenge, we need to find the indices in two equally sized lists where the numbers match. We will be iterating through both of them at the same time and comparing the values, if the numbers are equal, then we record the index. These are the steps we need to accomplish this:

1. Define our function to accept two lists of numbers
2. Create a new list to store our matching indices
3. Loop through each index to the end of either of our lists
4. Within the loop, check if our first list at the current index is equal to the second list at the current index. If so, append the index where they matched
5. Return our list of indices

Write a function named `same_values()` that takes two lists of numbers of equal size as parameters.

The function should return a list of the indices where the values were equal in `lst1` and `lst2`.

For example, the following code should return `[0, 2, 3]`

```
same_values([5, 1, -10, 3, 3], [5, 10, -10, 3, 5])
```

▼ **Hint**

Loop through all of the indices of each list using `for index in range(len(lst1))` and compare `lst1[index]` to `lst2[index]`. Append `index` to a new list if those two items are equal.

```
#Write your function here
def same_values(lst1, lst2):
  matching_indexes = []
  for i in range(len(lst1)):
    if lst1[i] == lst2[i]:
      matching_indexes.append(i)
  return matching_indexes

#Uncomment the line below when your function is done
print(same_values([5, 1, -10, 3, 3], [5, 10, -10, 3, 5]))
```

```
[0, 2, 3]
```

Here's how we did it:

```
def same_values(lst1, lst2):
  new_lst = []
  for index in range(len(lst1)):
    if lst1[index] == lst2[index]:
      new_lst.append(index)
  return new_lst
```

In this solution, we used a loop that iterates using the `range` of the `len` of our list. This generates the indices we need to iterate through. Note that we assume the lists are of equal size. We then access the element at the current index from each list using `lst1[index]` and `lst2[index]`. If they are equal we add the index to the new list. Finally, we return the results.

5. Reversed List

For the final challenge, we are going to test two lists to see if the second list is the reverse of the first list. There are a few different ways to approach this, but we are going to try a method that iterates through each of the values in one direction for the first list and compares them against the values starting from the other direction in the second list. Here is what you need to do:

1. Define a function that has two input parameters for our lists
2. Loop through every index in one of the lists from beginning to end
3. Within the loop, compare the element in the first list at the current index against the element at the second list's last index minus the current index. If there was a mismatch, then the lists aren't reversed and we can return `False`
4. If the loop ended successfully, then we know the lists are reversed and we can return `True`.

Create a function named `reversed_list()` that takes two lists of the same size as parameters named `lst1` and `lst2`.

The function should return `True` if `lst1` is the same as `lst2` reversed. The function should return `False` otherwise.

For example, `reversed_list([1, 2, 3], [3, 2, 1])` should return `True`.

▼ **Hint**

Let's say the lists are of size 5. You want to compare `lst1[0]` with `lst2[4]`, `lst1[1]` with `lst2[3]` and so on.
Loop through the numbers created by `range(len(lst1))` using a variable named `index`

Compare `lst1[index]` to `lst2[len(lst2) - 1 - index]`. If those two items are not equal, return `False`. If you loop through the entire list and you never return `False`, that means that every item was equal, and you should return `True`.

```
1   #Write your function here
2   def reversed_list(lst1, lst2):
3     reversed_index = len(lst2) -1
4     counting_matches = 0
5     for i in range(len(lst1)):
6       if lst1[i] == lst2[reversed_index]:
7         counting_matches += 1
8         reversed_index -= 1
9     if counting_matches == len(lst2):
10      return True
11    else:
12      return False
13
14  #Uncomment the lines below when your
    function is done
15  print(reversed_list([1, 2, 3], [3, 2, 1]))
16  print(reversed_list([1, 5, 3], [3, 2, 1]))
```

```
True
False
```

Here is one way to do it:

```
def reversed_list(lst1, lst2):
  for index in range(len(lst1)):
    if lst1[index] != lst2[len(lst2) - 1 - index]:
      return False
  return True
```

In this code, we iterate through each of the indices for the entire length of either of the lists (since we assume the lengths are equal) and we perform a comparison on each of the elements. We get the element at the current index from our first list with `lst1[index]` and we test it against the last index of the second list minus the current index `len(lst2) - 1 - index`.

That math is a little complicated — it helps to look at a concrete example. If we are given a list of 5 elements, the valid indices are 0 to 4. Because of this, the last index in

the second list is `len(lst2) - 1`, or `5 - 1 = 4`. Now in order to get the inverse of the position we are at in the first list, we subtract the index we are at from the end of the second list. So on the first pass, we'll compare the element at position `0` to the element at position `5 - 1 - 0 = 4`. On the next pass, we'll compare the element at position `1` to the element at position `5 - 1 - 1 = 3`, and so on.

If any of the two elements are not equal then we know that the second list is not the reverse of the first list and we return `False`. If we made it to the end without a mismatch then we can return `True` since the second list is the reverse of the first. You could also try simplifying this code by using the python function `reversed()` or other methods that you will learn later on such as 'slicing'.