Python Code Challenges: Loops

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. Divisible By Ten

Let's start our code challenges with a function that counts how many numbers are divisible by ten from a list of numbers. This function will accept a list of numbers as an input parameter and use a loop to check each of the numbers in the list. Every time a number is divisible by 10, a counter will be incremented and the final count will be returned. These are the steps we need to complete this:

1. Define the function to accept one input parameter called `nums`
2. Initialize a counter to 0
3. Loop through every number in `nums`
4. Within the loop, if any of the numbers are divisible by 10, increment our counter
5. Return the final counter value

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

Return the count of how many numbers in the list are divisible by 10.

▶ Hint

```
#Write your function here
def divisible_by_ten(nums):
  counter = 0
  for num in nums:
    if num % 10 == 0:
      counter += 1
  return counter

#Uncomment the line below when your function is done
print(divisible_by_ten([20, 25, 30, 35, 40]))
```

```
3
```

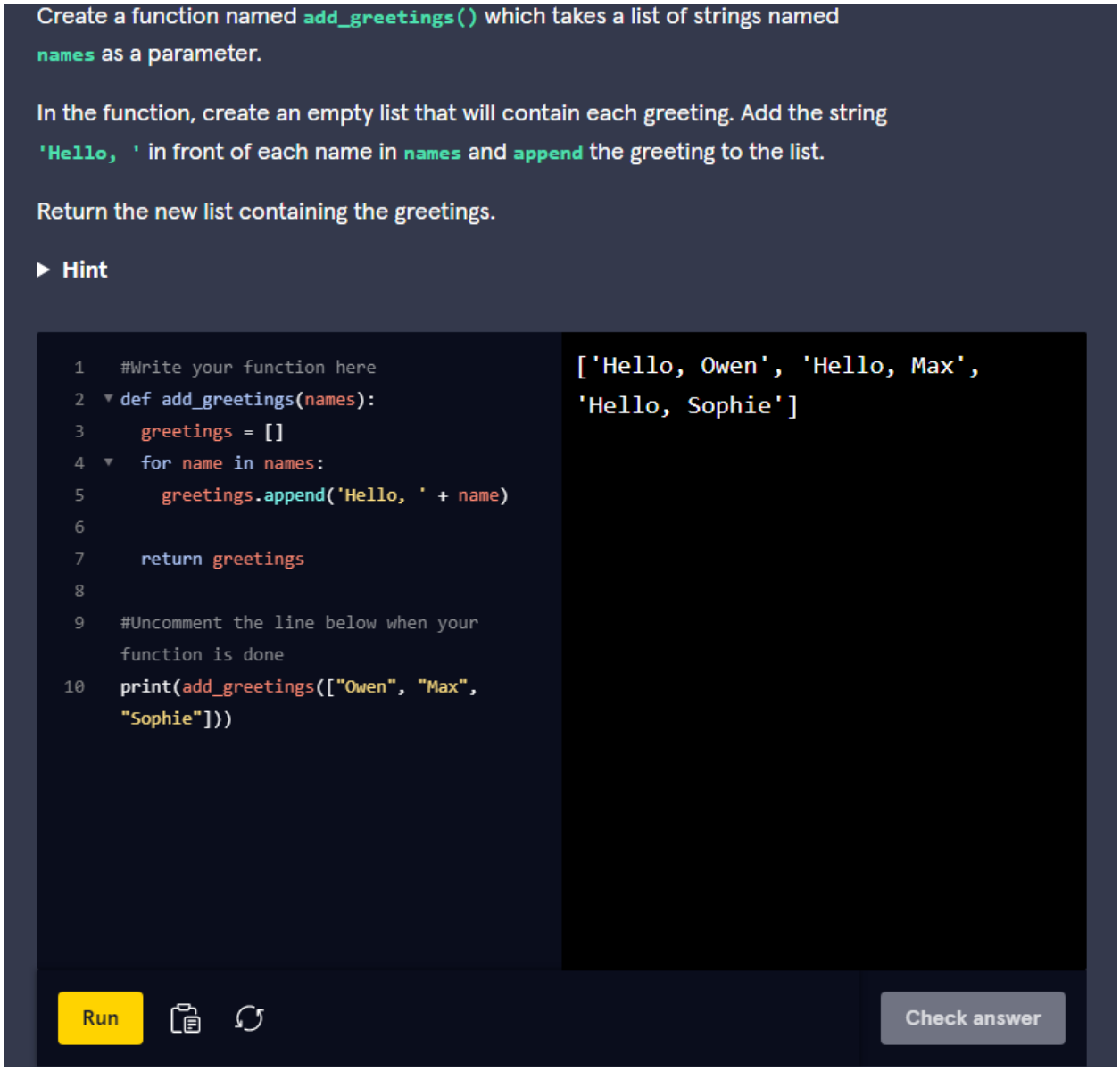
Run

Check answer

2. Greetings

You are invited to a social gathering, but you are tired of greeting everyone there. Luckily we can create a function to accomplish this task for us. In this challenge, we will take a list of names and prepend the string `'Hello, '` before each name. This will require a few steps:

1. Define the function to accept a list of strings as a single parameter called `names`
2. Create a new list of strings
3. Loop through each name in `names`
4. Within the loop, concatenate `'Hello, '` and the current name together and append this new string to the new list of strings
5. Afterwards, return the new list of strings



This is one way to solve it:

```
def add_greetings(names):
  new_list = []
  for name in names:
    new_list.append("Hello, " + name)
  return new_list
```

First, we set up our function to accept the list of strings and we initialized a new list of strings to hold our greetings. We iterate through each name and we append and concatenate the strings at the same time within our loop. Finally, we return the list containing all of our eloquent greetings.

3. Delete Starting Even Numbers

Let's try a tricky challenge involving removing elements from a list. This function will repeatedly remove the first element of a list until it finds an odd number or runs out of elements. It will accept a list of numbers as an input parameter and return the modified list where any even numbers at the beginning of the list are removed. To do this, we will need the following steps:

1. Define our function to accept a single input parameter `lst` which is a list of numbers
2. Loop through every number in the list if there are still numbers in the list and if we haven't hit an odd number yet
3. Within the loop, if the first number in the list is even, then remove the first number of the list
4. Once we hit an odd number or we run out of numbers, return the modified list

Write a function called `delete_starting_evens()` that has a parameter named `lst`.

The function should remove elements from the front of `lst` until the front of the list is not even. The function should then return `lst`.

For example if `lst` started as `[4, 8, 10, 11, 12, 15]`, then `delete_starting_evens(lst)` should return `[11, 12, 15]`.

Make sure your function works even if every element in the list is even!

▼ **Hint**

Use a while loop to check two things. First, check if the list has at least one element, using `len(lst)`. Second, check to see if the first element is odd using mod (`%`). If both of those are `True`, slice off the first element of the list using `lst = lst[1:]`.

```
#Write your function here
def delete_starting_evens(lst):
  longitud = len(lst)
  while longitud > 0:
    if lst[0] % 2 == 0:
      lst.remove(lst[0])
    longitud -= 1
  return lst



#Uncomment the lines below when your function is done
print(delete_starting_evens([4, 8, 10, 11, 12, 15]))
#print(delete_starting_evens([4, 8, 10]))
```

```
[11, 12, 15]
```

This is the way we solved it:

```
def delete_starting_evens(lst):
  while (len(lst) > 0 and lst[0] % 2 == 0):
    lst = lst[1:]
  return lst
```

After defining our method, we use a `while` loop to keep iterating as long as some provided conditions are true. We provide two conditions for the `while` loop to continue. We will keep iterating as long as there is at least one number left in the list `len(lst) > 0` and if the first element in the list is even `lst[0] % 2 == 0`. If both of these conditions are true, then we replace the list with every element except for the first one using `lst[1:]`. Once the list is empty or we hit an odd number, the `while` loop terminates and we return the modified list.

4. Odd Indices

This next function will give us the values from a list at every odd index. We will need to accept a list of numbers as an input parameter and loop through the odd indices instead of the elements. Here are the steps needed:

1. Define the function header to accept one input which will be our list of numbers
2. Create a new list which will hold our values to return
3. Iterate through every odd index until the end of the list
4. Within the loop, get the element at the current odd index and append it to our new list
5. Return the list of elements which we got from the odd indices.

Create a function named `odd_indices()` that has one parameter named `lst`.

The function should create a new empty list and add every element from `lst` that has an odd index. The function should then return this new list.

For example, `odd_indices([4, 3, 7, 10, 11, -2])` should return the list `[3, 10, -2]`.

▼ **Hint**

There are a few ways to do this. `range(1, len(lst), 2)` will create a list of the indices you're interested in. So you could loop through that list like this:

```
for index in range(1, len(lst), 2):
  new_list.append(lst[index])
```

```
#Write your function here
def odd_indices(lst):
  new_list = []
  for num in lst:
    if lst.index(num) % 2 != 0:
      new_list.append(num)
  return new_list

#Uncomment the line below when your function is done
print(odd_indices([4, 3, 7, 10, 11, -2]))
```

```
[3, 10, -2]
```

Here is this solution:

```
def odd_indices(lst):
  new_lst = []
  for index in range(1, len(lst), 2):
    new_lst.append(lst[index])
  return new_lst
```

In our solution, we iterate through a range of values. The function `range(1, len(lst), 2)` returns a list of numbers starting at 1, ending at the length of `len`, and incrementing by 2. This causes the loop to iterate through every odd number until the last index of our list of numbers. Using this, we can simply append the element at whatever index we are at since we know that using our `range` we will be iterating through only odd indices.

Another way to do this would be to iterate through all indices and use an if statement to see if the index you're currently looking at is odd.

5. Exponents

In this challenge, we will be using nested loops in order to raise a list of numbers to the power of a list of other numbers. What this means is that for every number in the first list, we will raise that number to the power of every number in the second list. If you provide the first list with 2 elements and the second list with 3 numbers, then there will be 6 final answers. Let's look at the steps we need:

1. Define the function to accept two lists of numbers, `bases` and `powers`
2. Create a new list that will contain our answers
3. Create a loop that iterates through every base in `bases`
4. Within that loop, create another loop that iterates through every power in `power`
5. Within that nested loop, append the result of the current base raised to the current power.
6. After all iterations of both loops are complete, return the list of answers

Here is how we solved this one:

```
def exponents(bases, powers):
  new_lst = []
  for base in bases:
    for power in powers:
      new_lst.append(base ** power)
  return new_lst
```

As you can see in this solution, we used two nested `for` loops so that, for every base, we iterate through every power. This allows us to raise each base to every single power in our list and append the answers to our new list. Finally, we return the list of answers.