# Python Code Challenges: Control Flow (Advanced)

**Difficult Python Code Challenges Involving Control Flow**

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

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

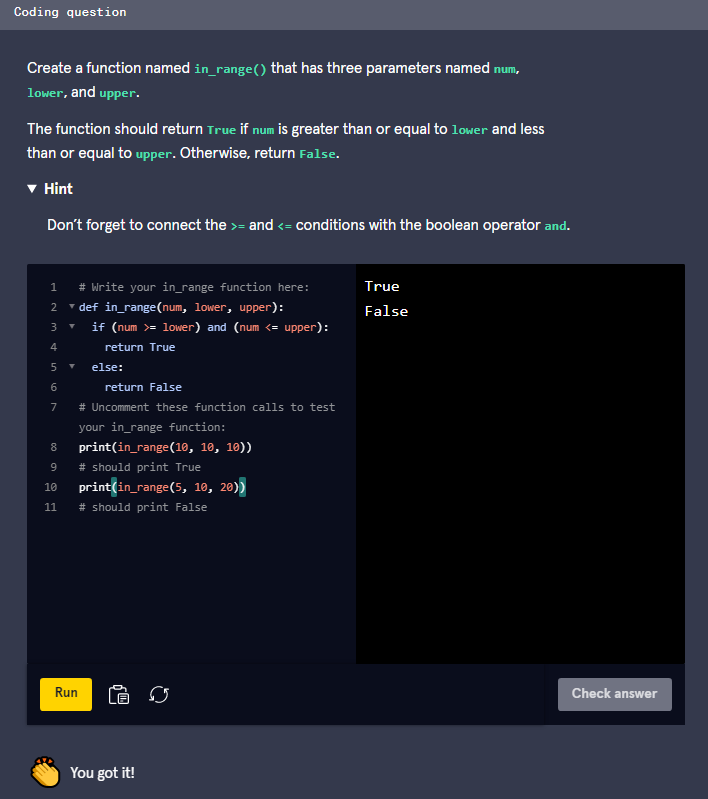
### Challenges

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

### 1. In Range

Let’s start the advanced challenge problems by testing if a number falls within a certain range. We will accept three parameters where the first parameter is the number we are testing, the second parameter is the lower bound and the third parameter is the upper bound of our range. These are the steps required:

1. Define the function to accept three numbers as parameters
2. Test if the number is greater than or equal to the lower bound and less than or equal to the upper bound
3. If this is true, return **True**, otherwise, return **False**



Here’s one way to do it:

def in\_range(num, lower, upper):  
  if(num >= lower and num <= upper):  
    return True  
  return False

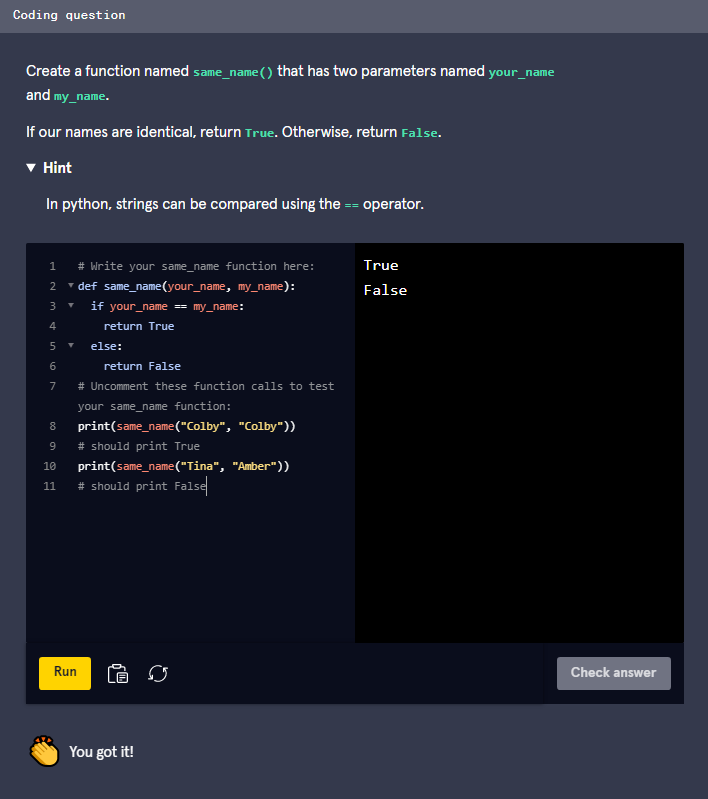
In this solution, we test the two bounds connected with an **and** boolean operator. This means that the code nested in the **if** statement will only execute if both of the conditions are true. We also do not include the **else** statement here. Since our **if** statement will return **True** and exit the function if the condition is true, the last line will only be reached if the condition was false.

**2. Same Name**

We need to write a program that checks different names and determines if they are equal. We need to accept two strings and compare them. Here are the steps:

1. Define the function to accept two strings, **your\_name** and **my\_name**
2. Test if the two strings are equal
3. Return **True** if they are equal, otherwise return **False**

Coding question



Here is this solution:

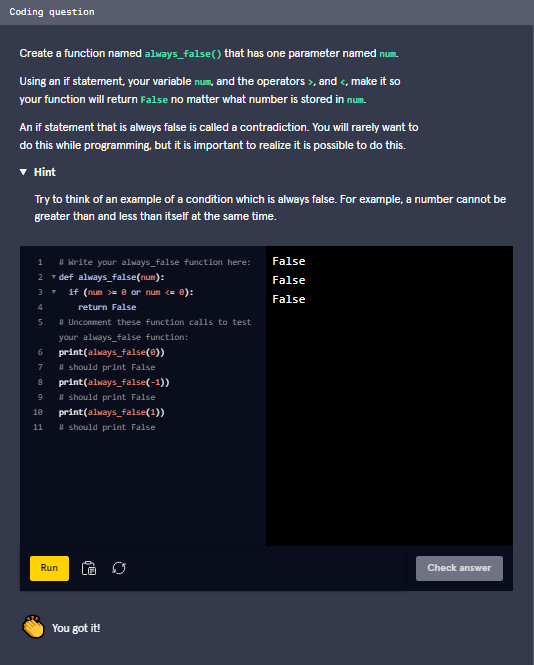
def same\_name(your\_name, my\_name):  
  if (your\_name == my\_name):  
    return True  
  else:  
    return False

As you can see in this solution code, comparing two strings in python can be done using the **==** operator. If you want an added challenge, you can try shortening the function body to one line of code!

### 3. Always False

There are some situations that you normally want to avoid when programming using conditional statements. One example is a contradiction. This occurs when your condition will always be false no matter what value you pass into it. Let’s create an example of a function that contains a contradiction. It will contain a few steps:

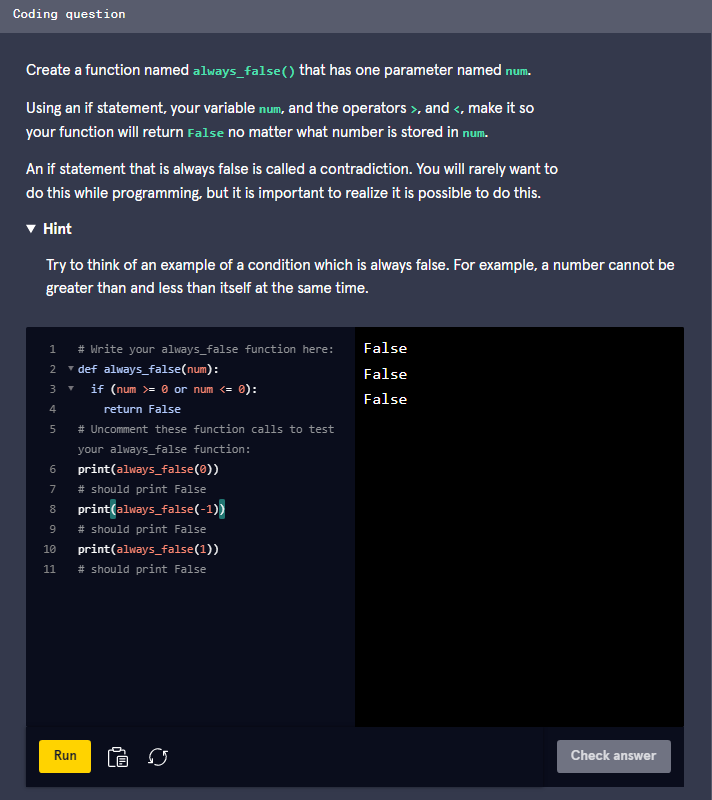
1. Define the function to accept a single parameter called **num**
2. Use a combination of **<**, **>** and **and** to create a contradiction in an **if** statement.
3. If the condition is true, return **True**, otherwise return **False**. The trick here is that because we’ve written a contradiction, the condition should never be true, so we should expect to always return **False**.



### 3. Always False

There are some situations that you normally want to avoid when programming using conditional statements. One example is a contradiction. This occurs when your condition will always be false no matter what value you pass into it. Let’s create an example of a function that contains a contradiction. It will contain a few steps:

1. Define the function to accept a single parameter called **num**
2. Use a combination of **<**, **>** and **and** to create a contradiction in an **if** statement.
3. If the condition is true, return **True**, otherwise return **False**. The trick here is that because we’ve written a contradiction, the condition should never be true, so we should expect to always return **False**.



Here is one way to solve this:

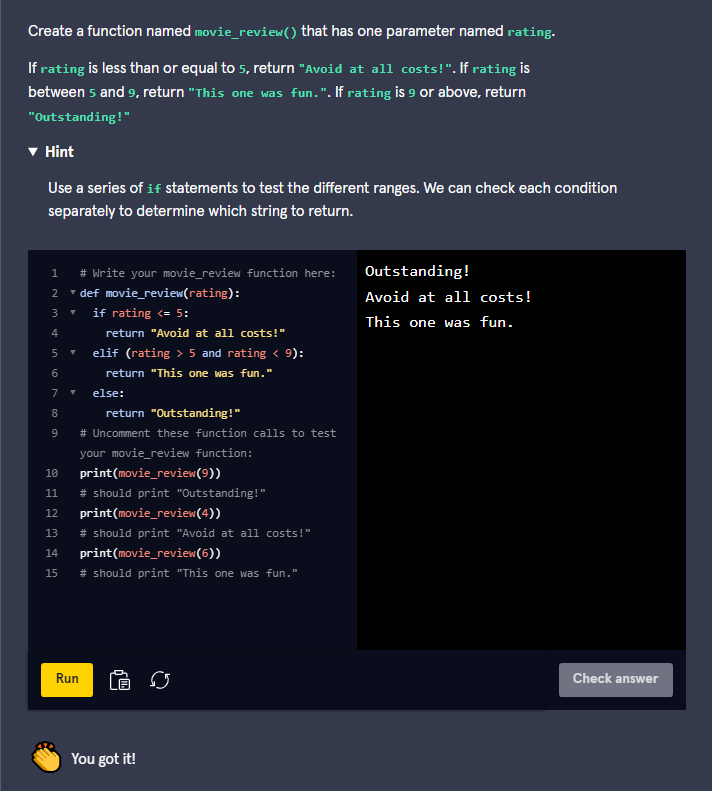
def always\_false(num):  
  if (num > 0 and num < 0):  
    return True  
  else:  
    return False

In our example, we use the contradiction of being greater than and less than 0 at the same time. No matter what value we pass into the function, our condition will always be false since it is not logically possible. You normally want to avoid creating conditions like this.

### 4. Movie Review

We want to create a function that will help us rate movies. Our function will split the ratings into different ranges and tell the user how the movie was based on the movie’s rating. Here are the steps needed:

1. Define our function to accept a single number called **rating**
2. If the rating is equal to or less than 5, return **"Avoid at all costs!"**
3. If the rating was less than 9, return **"This one was fun."**
4. If neither of the **if** statement conditions were met, return **"Outstanding!"**



Here’s how we did it:

def movie\_review(rating):  
  if(rating <= 5):  
    return "Avoid at all costs!"  
  if(rating < 9):  
    return "This one was fun."  
  return "Outstanding!"

To solve this, we used a series of **if** statements to select which string to return. Another way of solving this would be to use **if**, **elif** and **else** statements.

### 5. Max Number

For the final challenge, we are going to select which number from three input values is the greatest using conditional statements. To do this, we need to check the different combinations of values to see which number is greater than the other two. Here is what we need to do:

1. Define a function that has three input parameters, **num1**, **num2**, and **num3**
2. Test if **num1** is greater than the other two numbers
   * If so, return **num1**
3. Test if **num2** is greater than the other two numbers
   * If so, return **num2**
4. Test if **num3** is greater than the other two numbers
   * If so, return **num3**
5. If there was a tie between the two largest numbers, then return **"It's a tie!"**

