

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`

Coding question

Create a function named `in_range()` that has three parameters named `num`, `lower`, and `upper`.

The function should return `True` if `num` is greater than or equal to `lower` and less than or equal to `upper`. Otherwise, return `False`.

▼ Hint

Don't forget to connect the `>=` and `<=` conditions with the boolean operator `and`.

```
1 # Write your in_range function here:
2 def in_range(num, lower, upper):
3     if (num >= lower) and (num <= upper):
4         return True
5     else:
6         return False
7 # Uncomment these function calls to test
  your in_range function:
8 print(in_range(10, 10, 10))
9 # should print True
10 print(in_range(5, 10, 20))
11 # should print False
```

True
False

Run



Check answer



You got it!

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

Coding question

Create a function named `same_name()` that has two parameters named `your_name` and `my_name`.

If our names are identical, return `True`. Otherwise, return `False`.

▼ Hint

In python, strings can be compared using the `==` operator.

```
1 # Write your same_name function here:
2 def same_name(your_name, my_name):
3     if your_name == my_name:
4         return True
5     else:
6         return False
7 # Uncomment these function calls to test
  your same_name function:
8 print(same_name("Colby", "Colby"))
9 # should print True
10 print(same_name("Tina", "Amber"))
11 # should print False
```

True
False

Run



Check answer



You got it!

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`.

Coding question

Create a function named `always_false()` that has one parameter named `num`.

Using an if statement, your variable `num`, and the operators `>`, and `<`, make it so your function will return `False` no matter what number is stored in `num`.



An if statement that is always false is called a contradiction. You will rarely want to do this while programming, but it is important to realize it is possible to do this.


▼ Hint

Try to think of an example of a condition which is always false. For example, a number cannot be greater than and less than itself at the same time.

```
1 # Write your always_false function here:
2 def always_false(num):
3     if (num >= 0 or num <= 0):
4         return False
5 # Uncomment these function calls to test
  your always_false function:
6 print(always_false(0))
7 # should print False
8 print(always_false(-1))
9 # should print False
10 print(always_false(1))
11 # should print False
```

False
False
False

Run   Check answer

 You got it!

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**.

Coding question

Create a function named `always_false()` that has one parameter named `num`.

Using an if statement, your variable `num`, and the operators `>`, and `<`, make it so your function will return **False** no matter what number is stored in `num`.

An if statement that is always false is called a contradiction. You will rarely want to do this while programming, but it is important to realize it is possible to do this.

▼ Hint

Try to think of an example of a condition which is always false. For example, a number cannot be greater than and less than itself at the same time.

```
1 # Write your always_false function here:
2 def always_false(num):
3     if (num >= 0 or num <= 0):
4         return False
5 # Uncomment these function calls to test
  your always_false function:
6 print(always_false(0))
7 # should print False
8 print(always_false(-1))
9 # should print False
10 print(always_false(1))
11 # should print False
```

False
False
False

Run



Check answer

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!"

Create a function named `movie_review()` that has one parameter named `rating`.

If `rating` is less than or equal to 5, return "Avoid at all costs!". If `rating` is between 5 and 9, return "This one was fun.". If `rating` is 9 or above, return "Outstanding!"

▼ Hint

Use a series of `if` statements to test the different ranges. We can check each condition separately to determine which string to return.

```
1 # Write your movie_review function here:
2 def movie_review(rating):
3     if rating <= 5:
4         return "Avoid at all costs!"
5     elif (rating > 5 and rating < 9):
6         return "This one was fun."
7     else:
8         return "Outstanding!"
9     # Uncomment these function calls to test
   your movie_review function:
10 print(movie_review(9))
11 # should print "Outstanding!"
12 print(movie_review(4))
13 # should print "Avoid at all costs!"
14 print(movie_review(6))
15 # should print "This one was fun."
```

Outstanding!
Avoid at all costs!
This one was fun.

Run



Check answer



You got it!

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!"**

Create a function called `max_num()` that has three parameters named `num1`, `num2`, and `num3`.

The function should return the largest of these three numbers. If any of two numbers tie as the largest, you should return `"It's a tie!"`.

▼ Hint

You can use `if`, `elif`, and `else` statements to accomplish this. For the first parameter, test if `num1` is greater than `num2` and that `num1` is greater than `num3`. Repeat these tests for `num2` and `num3`.

```
1 # Write your max_num function here:
2 def max_num(num1, num2, num3):
3     if (num1 > num2 and num1 > num3):
4         return num1
5     if (num2 > num1 and num2 > num3):
6         return num2
7     if (num3 > num1 and num3 > num2):
8         return num3
9     if (num1 == num2 or num1 == num3):
10        return "It's a tie!"
11    if (num2 == num1 or num2 == num3):
12        return "It's a tie!"
13    if (num3 == num1 or num3 == num2):
14        return "It's a tie!"
15    # Uncomment these function calls to test
    # your max_num function:
16    print(max_num(-10, 0, 10))
17    # should print 10
18    print(max_num(-10, 5, -30))
--
```

```
10
5
-5
It's a tie!
```

Run



Check answer



You got it!

Here is one way to do it:

```
def max_num(num1, num2, num3):
    if num1 > num2 and num1 > num3:
        return num1
    elif num2 > num1 and num2 > num3:
        return num2
    elif num3 > num1 and num3 > num2:
        return num3
    else:
        return "It's a tie!"
```

In this code, we use a series of `if`, `elif`, and `else` statements. We test the first parameter against the other two parameters and return the value if it is greater than the other two. We have two more tests to check if the second parameter is greater than the other two, then if the third parameter is greater than the other two. In the case where none of the parameters were greater than both of the other parameters, then we know that there must have been a tie and the final return statement is reached.

