The if Statement

This lesson showcases the functionality of the 'if' statement.

We'll cover the following

The Structure

The simplest conditional statement that we can write is the `if` statement. It comprises of two parts:

1. The **condition**
2. The **code to be executed**

The `:` in the illustration above is necessary to specify the beginning of the `if` statement's code to be executed. However, the parentheses, `()`, around the condition are optional. The code to be executed is indented at least one tab to the right.

Indentation

Indentation plays an essential role in Python. Statements with the same level of indentation belong to the same block of code. The code of an `if` statement is indented a space further than the code outside it in order to indicate that this is an *inner* and *inter-related* block.

The convention of our indents must also be consistent throughout a block. If we have used two spaces to make an indent, we must use two spaces for an indent in the same block. Hence, always keep indentation in mind when writing code.

We'll see later on how indents play a role in other aspects of Python.

The Flow of an `if` Statement

An `if` statement runs like this:

> **if** the **condition** holds `True`, execute the **code to be executed**. Otherwise, **skip** it and move on.

Let's write a simple `if` statement that verifies the value of an integer:

```
num = 5

if (num == 5):  # The condition is true
    print("The number is equal to 5")  # The code is executed

if num > 5:  # The condtion is false
    print("The number is greater than 5")  # The code is not executed
```

Our first condition simply checks whether the value of `num` is `5`. Since this Boolean expression returns `True`, the compiler goes ahead and executes the `print` statement on **line 4**.

As we can see, the `print` command inside the body of the `if` statement is indented to the right. If it wasn't, there would be an error. Python puts a lot of emphasis on proper indentation.

Conditions with Logical Operators

We can use logical operators to create more complex conditions in the `if` statement. For example, we may want to satisfy multiple clauses for the expression to be `True`.

```
num = 12

if num % 2 == 0 and num % 3 == 0 and num % 4 == 0:
    # Only works when num is a multiple of 2, 3, and 4
    print("The number is a multiple of 2, 3, and 4")

if (num % 5 == 0 or num % 6 == 0):
    # Only works when num is either a multiple of 5 or 6
    print("The number is a multiple of 5 and/or 6")
```

In the first `if` statement, all the conditions have to be fulfilled since we're using the `and` operator.

In the second `if` statement, the Boolean expression would be true if either of the clauses are satisfied because we are using the `or` operator.

Nested `if` Statements

A cool feature of conditional statements is that we can nest them. This means that there could be an `if` statement inside another!

Hence, we can use nesting to make complex conditions in our program:

```
num = 63

if num >= 0 and num <= 100:
    if num >= 50 and num <= 75:
        if num >= 60 and num <= 70:
            print("The number is in the 60-70 range")
```

Note: Each nest `if` statement requires further indentation.

Creating and Editing Values

In a conditional statement, we can edit the values of our variables.

Furthermore, we can create new variables.

```
num = 10
if num > 5:
    num = 20  # Assigning a new value to num
    new_num = num * 5  # Creating a new value called newNum

# The if condition ends, but the changes made inside it remain
print(num)
print(new_num)
```

The `if` statement is the foundation of conditional programming in Python. The next two types of conditional statements we'll learn are simply extensions of `if`.

In the next lesson, we'll cover the `if-else` statement.

← Back

Next →

What are Conditional Statements?

The if-else Statement

☑ Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/the-if-statement__conditional-statements__learn-python-3-from-scratch)