Theory: If statement

14 minutes 10 / 10 problems solved

Start practicing

§1. Simple if statement

There are situations when your program needs to execute some piece of the code only if a particular condition is true. Such a piece of the code should be placed within the body of an **if statement**. The pattern is the same as in the English language: first comes the keyword `if` , then a condition, and then a list of expressions to execute. The condition is always a **Boolean expression**, that is, its value equals either `True` or `False`. Here is one example of how the code with a conditional expression should look like:

```
biscuits = 17
if biscuits >= 5:
    print("It's time for tea!")
```

Note that the condition ends with a colon and a new line starts with an **indentation**. Usually, 4 spaces are used to designate each level of indentation. A piece of code in which all lines are on the same level of indentation is called a **block of code**. In Python, *only* indentation is used to separate different blocks of code, hence, only indentation shows which lines of code are supposed to be executed when the `if` statement is satisfied, and which ones should be executed *independently* of the `if` statement. Check out the following example:

```
if biscuits >= 5:
    print("It's time for tea!")
    print("What tea do you prefer?")
print("What about some chocolate?")
```

In this example, the line `"It's time for tea!"`, as well as `"What tea do you prefer?"`, will be printed only if there are 5 or more biscuits. The line `"What about some chocolate?"` will be printed regardless of the number of biscuits.

An `if` statement is executed only if its condition holds (the Boolean value is `True`), otherwise, it's skipped.

Boolean values basically make it clear whether a piece of code needs to be executed or not. Since comparisons result in `bool`, it's always a good idea to use them as a condition.

There is one pitfall, though. You should not confuse the comparison operator for equality `==` with the assignment operator `=`. Only the former provides for a proper condition. Try to avoid this common mistake in your code.

§2. Nested if statement

Sometimes a condition happens to be too complicated for a simple `if` statement. In this case, you can use so-called **nested** `if` statements. The more `if` statements are nested, the more complex your code gets, which is usually not a good thing. However, this doesn't mean that you need to avoid nested `if` statements at all costs. Let's take a look at the code below:

```
rainbow = "red, orange, yellow, green, blue, indigo, violet"
warm_colors = "red, yellow, orange"
my_color = "orange"

if my_color in rainbow:
    print("Wow, your color is in the rainbow!")
    if my_color in warm_colors:
        print("Oh, by the way, it's a warm color.")
```

Current topic:
If statement Stage 1 16★

Topic depends on:
Comparisons Stage 1 16★

Topic is required for:
Else statement Stage 1 15★
While loop 11★
For loop Stage 1 12★
Search in a string 4★
Array creation and indexing

Table of contents:
↑ If statement
§1. Simple if statement
§2. Nested if statement
Feedback & Comments

The example above illustrates a nested `if` statement. If the variable `my_color` is a string that contains the name of a color from the rainbow, we enter the body of the first `if` statement. First, we print the message and then check if our color belongs to the warm colors. The membership operator `in` simply shows whether `my_color` is a substring of the respective string, `rainbow` or `warm_colors`. Just like arithmetic comparisons, it returns a boolean value.

Here is what we will see in our case:

```
Wow, your color is in the rainbow!
Oh, by the way, it's a warm color.
```

When it comes to nested `if` statements, proper indentation is crucial, so do not forget to indent each statement that starts with the `if` keyword.

Report a typo

1177 users liked this theory. **13** didn't like it. **What about you?**

Start practicing

Comments (13) Hints (3) Useful links (0) Show discussion