02-if, elif, and else Statements

October 29, 2020

Coursework delivered by: Alison Mukoma

0.1 Python vs Other Languages

Let's create a simple statement that says: "If a is greater than b, assign 2 to a and 4 to b"

Take a look at these two if statements (we will learn about building out if statements soon).

Version 1 (Other Languages)

```
if (a>b){
    a = 2;
    b = 4;
}
```

Version 2 (Python)

```
if a>b:
    a = 2
    b = 4
```

You'll notice that Python is less cluttered and much more readable than the first version. How does Python manage this?

Let's walk through the main differences:

Python gets rid of () and {} by incorporating two main factors: a *colon* and *whitespace*. The statement is ended with a colon, and whitespace is used (indentation) to describe what takes place in case of the statement.

Another major difference is the lack of semicolons in Python. Semicolons are used to denote statement endings in many other languages, but in Python, the end of a line is the same as the end of a statement.

Lastly, to end this brief overview of differences, let's take a closer look at indentation syntax in Python vs other languages:

0.2 Indentation

Here is some pseudo-code to indicate the use of whitespace and indentation in Python:

Other Languages

```
if (x)
    if(y)
        code-statement;
else
    another-code-statement;
```

Python

```
if x:
    if y:
        code-statement
else:
    another-code-statement
```

Note how Python is so heavily driven by code indentation and whitespace. This means that code readability is a core part of the design of the Python language.

Now let's start diving deeper by coding these sort of statements in Python!

0.3 Time to code!

if Statements in Python allows us to tell the computer to perform alternative actions based on a certain set of results.

Verbally, we can imagine we are telling the computer:

"Hey if this case happens, perform some action"

We can then expand the idea further with elif and else statements, which allow us to tell the computer:

"Hey if this case happens, perform some action. Else, if another case happens, perform some other action. Else, if *none* of the above cases happened, perform this action."

Let's go ahead and look at the syntax format for if statements to get a better idea of this:

```
if case1:
    perform action1
elif case2:
    perform action2
else:
    perform action3
```

0.4 First Example

Let's see a quick example of this:

[1]:
```
if True:
    print('It was true!')
```

```
It was true!
```

Let's add in some else logic:

[2]:
```
x = False

if x:
    print('x was True!')
else:
    print('I will be printed in any case where x is not true')
```

```
I will be printed in any case where x is not true
```

0.4.1 Multiple Branches

Let's get a fuller picture of how far if, elif, and else can take us!

We write this out in a nested structure. Take note of how the if, elif, and else line up in the code. This can help you see what if is related to what elif or else statements.

We'll reintroduce a comparison syntax for Python.

[3]:
```
loc = 'Bank'

if loc == 'Auto Shop':
    print('Welcome to the Auto Shop!')
elif loc == 'Bank':
    print('Welcome to the bank!')
else:
    print('Where are you?')
```

```
Welcome to the bank!
```

Note how the nested if statements are each checked until a True boolean causes the nested code below it to run. You should also note that you can put in as many elif statements as you want before you close off with an else.

Let's create two more simple examples for the if, elif, and else statements:

[4]:
```
person = 'Sammy'

if person == 'Sammy':
    print('Welcome Sammy!')
else:
    print("Welcome, what's your name?")
```

```
Welcome Sammy!
```

[5]:
```
person = 'George'

if person == 'Sammy':
    print('Welcome Sammy!')
elif person =='George':
    print('Welcome George!')
else:
    print("Welcome, what's your name?")
```

```
Welcome George!
```

0.5 Indentation

It is important to keep a good understanding of how indentation works in Python to maintain the structure and order of your code. We will touch on this topic again when we start building out functions!

0.6 Testing truth value

0.6.1 `bool`

[1]:
```
print('type of True and False: {}'.format(type(True)))
```

```
type of True and False: <class 'bool'>
```

[2]:
```
print('0: {}, 1: {}'.format(bool(0), bool(1)))
print('empty list: {}, list with values: {}'.format(bool([]), bool(['woop'])))
print('empty dict: {}, dict with values: {}'.format(bool({}), bool({'Python':␣
 ↪'cool'})))
```

```
0: False, 1: True
empty list: False, list with values: True
empty dict: False, dict with values: True
```

0.6.2 `==, !=, >, <, >=, <=`

[3]:
```
print('1 == 0: {}'.format(1 == 0))
print('1 != 0: {}'.format(1 != 0))
print('1 > 0: {}'.format(1 > 0))
print('1 > 1: {}'.format(1 > 1))
print('1 < 0: {}'.format(1 < 0))
print('1 < 1: {}'.format(1 < 1))
print('1 >= 0: {}'.format(1 >= 0))
print('1 >= 1: {}'.format(1 >= 1))
print('1 <= 0: {}'.format(1 <= 0))
print('1 <= 1: {}'.format(1 <= 1))
```

```
1 == 0: False
1 != 0: True
```

```
1 > 0: True
1 > 1: False
1 < 0: False
1 < 1: False
1 >= 0: True
1 >= 1: True
1 <= 0: False
1 <= 1: True
```

You can combine these:

[4]:
```
print('1 <= 2 <= 3: {}'.format(1 <= 2 <= 3))
```

```
1 <= 2 <= 3: True
```

0.6.3 and, or, not

[5]:
```
python_is_cool = True
java_is_cool = False
empty_list = []
secret_value = 3.14
```

[6]:
```
print('Python and java are both cool: {}'.format(python_is_cool and␣
 ↪java_is_cool))
print('secret_value and python_is_cool: {}'.format(secret_value and␣
 ↪python_is_cool))
```

```
Python and java are both cool: False
secret_value and python_is_cool: True
```

[7]:
```
print('Python or java is cool: {}'.format(python_is_cool or java_is_cool))
print('1 >= 1.1 or 2 < float("1.4"): {}'.format(1 >= 1.1 or 2 < float('1.4')))
```

```
Python or java is cool: True
1 >= 1.1 or 2 < float("1.4"): False
```

[8]:
```
print('Java is not cool: {}'.format(not java_is_cool))
```

```
Java is not cool: True
```

You can combine multiple statements, execution order is from left to right. You can control the execution order by using brackets.

[9]:
```
print(bool(not java_is_cool or secret_value and  python_is_cool or empty_list))
print(bool(not (java_is_cool or secret_value and  python_is_cool or␣
 ↪empty_list)))
```

```
True
False
```

0.7 if

[10]:
```
statement = True
if statement:
    print('statement is True')

if not statement:
    print('statement is not True')
```

```
statement is True
```

[11]:
```
empty_list = []
# With if and elif, conversion to `bool` is implicit
if empty_list:
    print('empty list will not evaluate to True')  # this won't be executed
```

[12]:
```
val = 3
if 0 <= val < 1 or val == 3:
    print('Value is positive and less than one or value is three')
```

```
Value is positive and less than one or value is three
```

0.8 if-else

[13]:
```
my_dict = {}
if my_dict:
    print('there is something in my dict')
else:
    print('my dict is empty :(')
```

```
my dict is empty :(
```

0.9 if-elif-else

[14]:
```
val = 88
if val >= 100:
    print('value is equal or greater than 100')
elif val > 10:
    print('value is greater than 10 but less than 100')
else:
    print('value is equal or less than 10')
```

```
value is greater than 10 but less than 100
```

You can have as many `elif` statements as you need. In addition, `else` at the end is not mandatory.

[15]:
```
greeting = 'Hello fellow Pythonista!'
language = 'Italian'
```

```
if language == 'Swedish':
    greeting = 'Hejsan!'
elif language == 'Finnish':
    greeting = 'Latua perkele!'
elif language == 'Spanish':
    greeting = 'Hola!'
elif language == 'German':
    greeting = 'Guten Tag!'

print(greeting)
```

```
Hello fellow Pythonista!
```

For more detailed overview about conditionals, check this tutorial from Real Python.