Module 3; Notebook 1 - Conditional statements & integer operators

January 4, 2022

1 Module 3; Notebook 1 - Conditional statements & integer operators

To round off the learning content, we'll be looking at integer operators and conditional (`if-else`) statements ready for the final project of the workshop!

1.1 Integer operators

In the previous section we looked at the data types and operators, now let's see how we use operators on the numerical data types - integers & floats.

Identical to mathmatics, Python has operators to deal with the most common mathematic operations - here they are:

[ ]:
```
# Addition
addition_sum = 4 + 3 #7
# Subtraction
subtraction_sum = 7 - 3 #4
# Multiplication
multiplication_sum = 4 * 3 #12
# Division
division_sum = 12 / 4 #3.0

print(addition_sum, subtraction_sum, multiplication_sum, division_sum) #7, 4,␣
 ↪12, 3.0
```

The `/` operator is ued for float division, hence the 3.0 value of the division sum above - to change this to integer division change the `/` operator to `//` and run the cell above again!

In regards to operators precedence, Python follows the acronym PEMDAS:

P for Parentheses, then

E for Exponents, then

MD for Multiplication and division, left to right, then

AS for Addition and subtraction, left to right

This means that multiplication and division carry the same precedence as does addition and subtraction. Let's check out an example to see these in play:

[ ]:
```
10 - ((7 / 2) * 3) + 1 # 10 - (3.5 * 3) + 1 = 0.5
```

Staying true to the PEMDAS acronym, Python executed the parentheses the first and finished with the addition and subtraction! If you need some time to play around with the operators, here's a code cell for you to fool around with:

[ ]:
```
# Use this cell to have a play around with the operators
```

1.2 Conditional statements

Conditional statements, or if-else statements, provide us a way to handle decisions. Conditionals perform different actions depending on whether a boolean (True or False) condition evaluates to either True or False. Let's explore a few examples together:

[ ]:
```
x = 1
# Defining an if-statement
if x == 1: # Can be read as if x equals 1 then...
    # Code within a conditional statement needs indenting
    print("X is 1") # Print statement if the pre-defined condition (if x does␣
 ↪equal 1) is True
```

[ ]:
```
x = 2
# Defining an if-else statement
if x == 1: # Can be read as if x equals 1 then... requires a colon at the end␣
 ↪of the statement
    # Code within a conditional statement needs indenting
    print("X is 1") # Print statement if the pre-defined condition is True
else: # Else statement to handle situations where x isn't 1
    # Code within a conditional statement needs indenting
    print("X is not 1")
```

1.2.1 Syntax

In terms of laying out a conditional statement, we usually follow the layout:

if `specfic condition`:

`code to be executed if the above condition is met`

elif `another specfic condition`:

`code to be executed if the above condition is met`

else:

`code to be executed if none of the specific conditions are met`

Note the indentation of the 'code' inside the conditional clauses, this is crucial and part of the Python syntax. All conditions must end with a colon as show in the examples above.

Not all conditional statements have all of the clauses shown above, like the first cell in this example where we check if x = 1.

Additional note - elif is short for else if

```
[ ]: #Fully fledge example
     x = 12
     if x / 5 == 0: # Can X be divided exactly by 5
         print("X is exactly divisble by 5")
     elif x / 2.3 == 0: # Can X be divided exactly by 2.3
         print("X is exactly divisble by 2.3")
     else:
         print("X not exactly divisible by 5 or 2.3")
```

Now we've looked at conditionals and comparing integers, let's through floats into the equation…

```
[ ]: x = 2.0
     y = 2
     if x == y:
         print("Yay")
         print(type(x), type(y))
```

As we can see, we can compare the value of a float to an integer - the example equates to True as casting `2.0` to an integer will yield 2!

1.2.2 Comparing strings & characters

Now we've explored conditional statements and comparing integers, let's take a look at strings!

```
[ ]: if "Hello" == "HeLlo":
         print("The same!")
     else:
         print("Not the same!")
```

As each string e.g. `Hello` can have any variations e.g. `HeLlo`, we need a way to be able compare whether strings are actually the same…

```
[ ]: first_word = "HeLlO"
     print(first_word.lower()) # Convert the string to lowercase!
```

```
[ ]: if "Hello".lower() == "HeLlo".lower():
         print("The same!")
     else:
         print("Not the same!")
```

We use the `.lower()` function on both the strings we are comparing to convert them both to lowercase format - this allows for easy comparison!

```
[ ]: if 'A' == 'a':
         print("The same!")
     else:
         print("Not the same!")
```

Comparing characters works the same way as strings in the sense that letters need to be converted to lower case in order for the conditional statement to evaluate to True.

Below are a few other examples to aid your understanding!

[ ]:
```python
input_char = '*'
if input_char == '*':
    print('*')
```

[ ]:
```python
input_char = '$'
if input_char == '£':
    print('£')
elif input_char == '$':
    print('$')
else:
    print('Something else')
```

1.2.3 The != operator in conditional statements

In order to test if a variable is not equal to a certain value we use the operator ! which can be read as not. This operator is particularly useful if you only need to hand one value in a different way to others.

[ ]:
```python
x = 5
if x != 5:
    print("The value is not 5")
else:
    print("x is 5")
```

[ ]:
```python
char_1 = 'a'
char_2 = 'b'
if char_1 != char_2:
    print("The characters are different")
```