Introduction

You can't learn to write Python, or any programming language for that matter without learning how to work with numbers. In almost every program you are going to write as a professional programmer, numbers will be involved. For example payroll systems, time-tracking, data science, data visualization, web development. In this lesson we're going to look at the various ways that Python handles numbers.

Integers

Integers are whole numbers, 1, 2, 3. No decimal points, or fractions. The Python interpreter allows you to add (+), subtract (-), multiply (*), and divide integers ().

In [5]:
```
#Addition
print(1 + 1)

#Subtraction
print(3 - 2)

# Multiplication
print(3 * 3)
```

```
2
1
9
```

In Python, you use two multiplication (**) symbols to signify exponents or 'to the power of', 'raised to the power of', or 'squared'.

In [6]:
```
# Exponents
print(4 ** 4)
```

```
256
```

Python follows the numerical order of operations, which means that when executing calculations, it does so in a particular order, usually, parentheses, exponents, multiplication/division, addition/subtraction, or PEMDAS. You can use parentheses to change the order of numerical calculation so that Python will execute your expression in the order you want.

In [7]:
```
# PEMDAS
print(5 + 6 * 7)
print(5 * 6 + 7)
print(5 * (6 + 7))
```

```
47
37
65
```

Floats

In Python, a number with a decimal point is called a **float**. You'll find a similar term in other programming languages. A float refers to the fact that you may find a decimial point at any position in a number, 1.0, 1.001, 10.01.

In [8]:
```
# Floats
print(0.1 + 0.1)
print(0.2 + 0.2)
print(0.3 * 0.4)
print(0.6 - 0.5)
```

```
0.2
0.4
0.12
0.09999999999999998
```

As you can see from the last print statement, you can sometimes get several decimal places returned in your calculation. This happens across most programming languages. Don't worry too much about this for the moment. We'll come back to it in later lessons.

Integers and Floats

In Python, when you divide two integers by one another you will be returned with a float.

In [9]:
```
# Integer division
print(6/2)
```

```
3.0
```

If you divide an integer with a float you will get a float returned.

In [10]:
```
# Integer, float division
print(6/3.0)
```

```
2.0
```

Python defaults to a float in any numerical operation that uses a float even if the result is a whole number (integer).

Using underscores in numbers

In Python, when writing long numbers you can use underscores (_).

In [11]:
```
# Underscores in numbers
seconds_in_year = 31_536_000
print(seconds_in_year)
```

```
31536000
```

When Python saves a number that uses underscores, the underscores are ignored, so 100 is the same as 1_00, which is the same as 10_0. This is the same for integers and floats.

Multiple assignment

You can assign values to variables in a single line of code instead of using multiple lines.

```
In [12]: # Assign values to variables in one line
         a, b, c = 1, 2, 3
         print(a,b,c)
         print(b)
```

```
1 2 3
2
```

This technique is most often used when initializing numbers, for example:

```
In [13]: player_one, player_two, player_three = 0, 0, 0
         print(player_one, player_two, player_three)
```

```
0 0 0
```

Constants

To create a constant you use all capital letters. A constant is similar to a variable but its value never changes.

```
In [14]: # Minutes in an hour
         MINS_IN_HOUR = 60
         print(MINS_IN_HOUR)
```

```
60
```

Exercises

1. What is your age? Write a series of calculations that result in your age being printed to the screen.
2. Think of a number that never changes. Create a constant to represent that number and print it to the screen