Basic Arithmetic

In [30]:

```
# Addition
2+1
```

Out[30]:

```
3
```

In [31]:

```
# Subtraction
2-1
```

Out[31]:

```
1
```

In [32]:

```
# Multiplication
2*2
```

Out[32]:

```
4
```

In [33]:

```
# Division
3/2
```

Out[33]:

```
1.5
```

In [34]:

```
# Floor Division
7//4
```

Out[34]:

```
1
```

7 divided by 4 equals 1.75 not 1!

The reason we get this result is because we are using "floor" division. The // operator (two forward slashes) truncates the decimal without rounding, and returns an integer result.

In [35]:

```
# Modulo
7%4
```

Out[35]:

```
3
```

4 goes into 7 once, with a remainder of 3. The % operator returns the remainder after division.

In [36]:

```
# Powers
2**3
```

Out[36]:

```
8
```

In [37]:

```
# Can also do roots this way
4**0.5
```

Out[37]:

```
2.0
```

In [38]:

```
# Order of Operations followed in Python
2 + 10 * 10 + 3
```

Out[38]:

```
105
```

In [39]:

```
# Can use parentheses to specify orders
(2+10) * (10+3)
```

Out[39]:

```
156
```

Variable Assignments

In [40]:

```
# Let's create an object called "a" and assign it the number 5
a = 7
```

Now if I call a in my Python script, Python will treat it as the number 7.

In [41]:

```
# Adding the objects
a+a
```

Out[41]:

```
14
```

In [42]:

```
# Reassignment
a = 10
```

In [43]:

```
# Check
a
```

Out[43]:

```
10
```

Yes! Python allows you to write over assigned variable names. We can also use the variables themselves when doing the reassignment. Here is an example of what I mean:

In [44]:

```
# Check
a
```

Out[44]:

```
10
```

In [45]:

```
# Use A to redefine A
a = a + a
```

In [46]:

```
# Check
a
```

Out[46]:

```
20
```

The names you use when creating these labels need to follow a few rules:

1. Names can not start with a number.
2. There can be no spaces in the name, use _ instead.
3. Can't use any of these symbols :'",<>/?|()!@#$%^&*~-+
4. It's considered best practice (PEP8) that names are lowercase.
5. Avoid using the characters 'l' (lowercase letter el), 'O' (uppercase letter oh),or 'I' (uppercase letter eye) as single character variable names.
6. Avoid using words that have special meaning in Python like "list" and "str"

Dynamic Typing

Python uses dynamic typing, meaning you can reassign variables to different data types. This makes Python very flexible in assigning data types; it differs from other languages that are statically typed.

In [47]:

```
my_dogs = 2
```

In [48]:

```
my_dogs
```

Out[48]:

```
2
```

In [49]:

```
my_dogs = ['Babon', 'Rolli']
```

In [50]:

```
my_dogs
```

Out[50]:

```
['Babon', 'Rolli']
```

Pros and Cons of Dynamic Typing

Pros of Dynamic Typing

- very easy to work with
- faster development time

Cons of Dynamic Typing

- may result in unexpected bugs!
- you need to be aware of type()

Determining variable type with type()

You can check what type of object is assigned to a variable using Python's built-in type() function. Common data types include:

- **int** (for integer)
- **float**
- **str** (for string)
- **list**

- **tuple**
- **dict** (for dictionary)
- **set**
- **bool** (for Boolean True/False)

In [51]:

```
type(a)
```

Out[51]:

```
int
```

In [52]:

```
a = [1,2]
```

In [53]:

```
type(a)
```

Out[53]:

```
list
```

Simple Exercise

This shows how variables make calculations more readable and easier to follow.

In [54]:

```
my_income = 100
tax_rate = 0.1
my_taxes = my_income * tax_rate
```

In [55]:

```
my_taxes
```

Out[55]:

```
10.0
```