Chapter 2 - Operators

Operators

Arithmetic Operators

Learning Objectives - Arithmetic Operators

- **Define the symbols for arithmetic operations**
- **Describe what happens when the `+` and `*` operators are used with strings**
- **Demonstrate how to increment or decrement a variable**
- **Identify the order of operations**

Addition

The Addition (+) Operator

The addition operator works as you would expect with numbers.

```
print(7 + 3)
```

You can also add two variables together.

```
a = 7
b = 3
c = a + b
print(c)
```

challenge

What happens if you:

- Make `a` of type float (e.g. `a = 7.0`)?
- Make `b` a negative number (e.g. `b = -3`)?
- Make `b` an explicitly positive number (e.g. `b = +3`)

Incrementing Variables

Incrementing Variables

Incrementing a variable means to change the value of a variable by a set amount. You will most often have a counting variable, which means you will increment by 1.

```
a = 0
a = a + 1
print(a)
```

How to Read `a = a + 1`

The variable `a` appears twice on the same line of code. But each instance of `a` refers to something different.

How to Read a = a + 1

The += Operator

Incrementing is a common task for programmers. Many programming languages have developed a shorthand for `a = a + 1` because of this. `a += 1` does the same thing as `a = a + 1`.

```
a = 0
b = 0
a = a + 1
b += 1
print(a)
print(b)
```

challenge

What happens if you:

- Change `b` such that `b += 2`?
- Change `b` such that `b += -1`?
- Change `b` such that `b -= 1`?

Type Casting

Type Casting

Type casting (or type conversion) is when you change the data type of a variable.

```
a = 3
print(type(a))
a = str(a)
print(type(a))
```

▼ **What does `type` mean?**
The `type` command returns the data type of the value stored in a variable. Python abbreviates these types: `int` is an integer, `float` is a floating point number, `str` is a string, and `bool` is a boolean.

`a` is initially an integer, but `str(a)` converted `a` into a string.

challenge

What happens if you:

- Convert `a` to a floating point number?
- Convert `a` to a boolean?

Why Type Cast?

Do you know why the code below will not work?

```
a = 5
b = "3"
print(a + b)
```

You cannot add a string to an integer. You can convert `b` to an integer to fix the problem.

```
a = 5
b = "3"
print(a + int(b))
```

Data read from the keyboard or a file is always stored as a string. If you want to use this data, you will need to know how to convert it to the proper data type.

String Concatenation

String Concatenation

String concatenation is the act of combining two strings together. This is done with the `+` operator.

```
a = "This is an "
b = "example string"
c = a + b
print(c)
```

Strings can also use the `+=` operator for concatenation.

```
a = "This is an "
b = "example string"
a += b
print(a)
```

challenge

What happens if you:

- Concatenate two strings without an extra space (e.g. `a = "This is an"`)?
- Add `3` to a string?
- Add `"3"` to a string?

Subtraction

Subtraction

```
a = 10
b = 3
c = a - b
print(c)
```

challenge

What happens if you:

- Change `b` to `-3`?
- Change `c` to `c = a - -b`?
- Change `b` to `3.0`?
- Change `b` to `False`?

▼ **Subtracting a Boolean?**

In Python, boolean value are more than just true and false. False has the numerical value of 0, while true has the numerical value of 1. This is why doing math with a boolean does not give you an error message.

The -= Operator

Decrementing is the opposite of incrementing. Like `+=`, there is a shorthand for decrementing a variable - `-=`.

```
a = 10
b = 3
a -= b
print(a)
```

▼ **Subtraction and Strings**

You might be able to concatenate strings with the + operator, but you cannot use the - operator with them.

Division

Division

Division in Python is done with the `/` operator

```
a = 25
b = 5
print(a / b)
```

challenge

What happens if you:

- Change `b` to `0`?
- Change `b` to `True`?
- Change `b` to `0.5`?
- Change the code to

```
a = 25
b = 5
a /= b
print(a)
```

▼ **Reminder**

- `/=` works similar to `+=` and `-=`
- `True` has the value of 1
- `False` has the value of 0

Floor Division

Normal division in Python always returns a float. If you want a whole number, use floor division (`//`). Floor division does not round up, nor round down. It removes the decimal value from the answer.

Floor Division

```
a = 5
b = 2
print(a // b)
```

challenge

What happens if you:

- Change `b` to `5.1`?

▼ **Why is there a decimal?**

If floor division is about returning a whole number, why is the output of the above code `0.0`? Floor division will always return the value of a whole number even if the data type is a float. That is, if floor division returns a float, then it will always be `.0` for the decimal value. If you really want an integer as the result of floor division, you can always type cast.

```
a = 5
b = 2
print(int(a // b))
```

Multiplication

Multiplication

Python uses the `*` operator for multiplication.

```
a = 5
b = 10
print (a * b)
```

challenge

What happens if you:

- Change `b` to `0.1`?
- Change `b` to `-3`?
- Change `b` to `True`?

▼ **Reminder**

- `*=` works similar to `+=` and `-=`
- `True` has the value of 1
- `False` has the value of 0

Multiplication & Strings

Python allows you to multiply a string by a number.

```
a = 3
b = "Hello!"
print(a * b)
```

challenge

What happens if you:

- Change `a` to `3.0`?
- Change `a` to `-3`?

Powers

Powers

Python uses the `**` operator for powers (or exponents). So `2 ** 2` would be two to the second power.

```
a = 2 ** 2
print(a)
```

challenge

What happens if you:

- Change `a` to `2 ** 0`?
- Change `a` to `2 ** -2`?
- Change `a` to `2 ** False`?

Square Root

The square root of 4 can be calculated as 4 raised to the power of $\frac{1}{2}$. In Python, this is written as `4 ** 0.5`.

▼ **The `sqrt` function**

Python does have a `sqrt` function, but it requires you to import the `math` library. Libraries will be covered in a later lesson.

```
import math

square_root = math.sqrt(4)
print(square_root)
```

```
square_root = 4 ** 0.5
print(square_root)
```

Order of Operations

Order of Operations

Python uses the PEMDAS method for determining order of operations.

P **P**arentheses
E **E**xponents - powers & square roots
MD **M**ultiplication & **D**ivision - left to right
AS **A**ddition & **S**ubtraction - left to right

PEMDAS

The code below should output `7.0`.

```
a = 2
b = 3
c = 4
result = 3 * a ** 3 / (b + 5) + c
print(result)
```

▼ **Explanation**

- The first step is to compute `b + 5` (which is `8`) because it is surrounded by parentheses.
- Next, calculate `a ** 3` (which is `8`) because it is an exponent.
- Next, do the multiplication and division going from left to right. `3 * 8` is `24`.
- `24` divided by `8` is `3.0` (remember, the `/` operator returns a float).
- Finally, add `3.0` and `4` together to get `7.0`.

challenge

Mental Math

- `5 + 7 - 10 * 3 / 0.5`

  ▼

  Solution

  - **Step 1:** 10 * 3 = 30
  - **Step 2:** 30 / 0.5 = 60.0
  - **Step 3:** 7 - 60.0 = -53.0
  - **Step 4:** 5 + -53.0 = -48.0
  - **Solution:** -48.0

- `(5 * 8) - 7 ** 2 - (-1 * 18)`

▼ Solution

- **Step 1:** 5 * 8 = 40
- **Step 2:** -1 * 18 = -18
- **Step 3:** 7 ** 2 = 49
- **Step 4:** 40 - 49 = -9
- **Step 5:** -9 - -18 = 9
- **Solution:** 9

- `9 / 3 + (100 ** 0.5) - 3`

▼ Solution

- **Step 1:** 100 ** 0.5 = 10
- **Step 2:** 9 / 3 = 3.0
- **Step 3:** 3.0 + 10 = 13.0
- **Step 4:** 13.0 - 3 = 10.0
- **Solution:** 10.0

Modulo

Modulo

Modulo is the mathematical operation that performs division but returns the remainder. The modulo operator is `%`. The modulo operation happens during the multiplication and division step of the order of operations.

Modulo

```
modulo = 5 % 2
print(modulo)
```

▼ **Order of Operations**

Modulo is treated like multiplication or division, and is performed in a left to right manner.

challenge

What happens if you:

- Change `modulo` to `5 % -2`?
- Change `modulo` to `5 % 0`?
- Change `modulo` to `5 % True`?