Introduction to Computer Programming

Arithmetic and Assignment Operators

Operators

In programming, **operators** are symbols/characters that represent a process or computation.

The values that an operator acts on are called **operands**.

Arithmetic Operators

Arithmetic operators (+, -, /, * ....) are used with numerical operands to perform common mathematical operations.

$**$	Exponent
/	Division
$*$	Multiplication
//	Floor division (round down to the next integer)
$\%$	Modulo `a % b = a - b * (a // b)` (remainder)
$+$	Addition
$-$	Subtraction

Operands can be literal values

Example

Use arithmetic operators to produce the expression:
$a = (3 - 1) * 4$

In [9]:
```python
a = (3-1)*4

print(a)
```

```
8
```

Operands can alternatively be variables

Example

$a = 6$
$b = 1.5$

Divide a by b and print the result

In [10]:
```
a = 6
b = 1.5

print(a/b)
```

```
4.0
```

Arithmetic Operators (in order of precedence)

$**$	**Exponent**
$/, *, //, \%$	**Division, multiplication, floor division, modulo** (evaluated left to right in code)
$+, -$	**Addition, subtraction** (evaluated left to right in code)

Example

Find:

$$\frac{6+2}{4^2} = 0.5$$

In [12]:
```
(6 + 2) / 4**2
```

Out[12]:
```
0.5
```

Arithmetic operators on strings - a word of warning!

Adding string (text) data using $+$ *connects* the two strings

String data is not numerical data, so the numerical characters are not summed algebraically

```
a = 2
b = '2'

print(a + a)
```

> 4

```
print(a + b)
```

> Error.
> Cannot add numerical and non-numerical value

```
print(b + b)
```

```
22
```

Other arithmetic operators cannot be used on non-numerical values.

Floating point error and arithmetic operators

For most decimal fractions, floating-point numbers contains small error due to the way that they are stored on the computer.

The difference between the true value and the stored value of the floating point number (the error) is typically in the order of 17 d.p.

When we do arithmetic using floating point numbers, this error compounds, which can cause unexpected results.

```
In [6]: format(3.3, '.17f')          # error is at least 1.8 x 10**-16

Out[6]: '3.29999999999999982'
```

```
In [7]: format(1.1, '.17f')          # error is at least 9 x 10**-17

Out[7]: '1.10000000000000009'
```

```
In [1]: print(3.3 - 1.1)
        print(3.3 / 1.1)
        print(3.3 + 1.1)

2.1999999999999997
2.9999999999999996
4.4
```

```
In [5]: format(3.3 + 1.1, '.17f')          # error is at least 3.6 x 10**-16

Out[5]: '4.40000000000000036'
```

In Python the result is rounded to either 15 or 17 significant figures depending on the computation being done

```
In [20]: 0.14 + 0.1

Out[20]: 0.24000000000000002
```

```
In [1]: 0.43 + 0.511

Out[1]: 0.9410000000000001
```

```
In [2]: 1.43 + 2.8

Out[2]: 4.2299999999999995
```

There are a number of ways to deal with this.

One of the simplest ways is to:

- select a degree of accuracy
- round the outcome of the operation to the required degree of accuracy

The Python function `round()` rounds a number to a specified number of decimal places.

In [6]:
```
dp = 10

a = 0.14 + 0.1
print(a)

a = round(0.14 + 0.1, dp)
print(a)
```

```
0.24000000000000002
0.24
```

The Python function `round()` rounds the first comma-seperated value within the parentheses to the number of decimal places represented by the second comma-seperated value within the parentheses

In [42]:
```
print(5e-324)

print(1e-324)

print(2e-324)
```

```
5e-324
0.0
0.0
```

Variable Reassignment

We can reassign the value of an existing variable

In [21]:
```
a = 1

a = 2

print(a)
```

```
2
```

The simplest assignment operator is the equals sign `=`.
`=` assigns value (object) to a variable.

It is valid for the value to the right of the assignment to include reference(s) to the variable that is being assigned to

In [22]:
```python
a = 1

a = a * 2 + 1

print(a)
```
```
3
```

Augmented Assignment Operators

Augmented assignment operators compound the `=` operator with any arithmetic operator (e.g. `+=`, `-=`) as a shortened expression to re-assign the value of a variable.

The new value of `a` is the original value of `a` when modified using the compounded arithmetic operator and value on the right hand side.

In [23]:
```python
a = 1
a += 5      # Equivalent to a = a + 5
print(a)
```
```
6
```

In [24]:
```python
a = 3

# Equivalent to a = a - 3
a -= 3

print(a)
```
```
0
```

In [25]:
```python
a = 4

# Equivalent to a = a /2
a /= 2

print(a)
```
```
2.0
```

Operators (in order of precedence)

1. Parentheses
2. Arithmetic operators (top to bottom)
   `**`   Exponent
   `/`, `*`, `//`, `\%`   Division, multiplication, floor division, modulo (evaluated left to right in code)
   `+`, `-`   Addition, subtraction (evaluated left to right in code)
3. Comparison operators: `<`, `<=`, `>`, `>=`, `!=`, `==` (evaluated left to right in code)
4. Assignment operators `=`, `/=`, `*=`, `//=`, `\%=`, `+=`, `-=` ....
5. Identity operators `is`, `is not`

6. Logical `not`
7. Logical `and`
8. Logical `or`

Need to see some more examples?

https://www.w3schools.com/python/python_operators.asp (https://www.w3schools.com/python/python_operators.asp)
https://www.geeksforgeeks.org/python-operators/ (https://www.geeksforgeeks.org/python-operators/)
https://www.programiz.com/python-programming/operators (https://www.programiz.com/python-programming/operators)
https://pynative.com/python-operators/ (https://pynative.com/python-operators/)

Want to take a quiz?

https://realpython.com/quizzes/python-operators-expressions/ (https://realpython.com/quizzes/python-operators-expressions/)
https://pynative.com/python-operators-and-expression-quiz/ (https://pynative.com/python-operators-and-expression-quiz/)

Want some more advanced information?

https://realpython.com/python-operators-expressions/ (https://realpython.com/python-operators-expressions/)

```
In [ ]:
```