Basic Python Syntax

Your code is stored in plain text files, which usually end on `.py`. More on files later.

First, here are some basics.

'#' Marks Comments

In [1]:

```
# I am a comment
```

In [3]:

```
print('Hello World!')  # prints a string
```

```
Hello World!
```

End-of-Line Terminates a Statement

The return key (↵) marks the end of a line.

In [4]:

```
print('Hello World!')
print('Next Statement')
```

```
Hello World!
Next Statement
```

So, programs really are just sequences of statements.

Expressions with Basic Operators

In [5]:

```
print(2 + 3 * 6)
```

```
20
```

Nå, I thought the result will be `30` ...

Why isn't it?

In [6]:

```
(2 + 3) * 6
```

Out[6]:

30

You can compute arbitrarily complex operations.

In [7]:

```
(5 - 1) * ((7 + 1) / (3 - 1))
```

Out[7]:

16.0

But what happened here?

In [ ]:

```
(4) * ((7 + 1) / (3 - 1))
```

In [ ]:

```
(4) * ((8) / (3 - 1))
```

In [ ]:

```
(4) * ((8) / (2))
```

In [ ]:

```
(4) * (4)
```

In [ ]:

```
16
```

Math operators from highest to lowest precedence:

Operator	Operation	Example	Evaluates to...
`**`	Exponent	`2 ** 3`	8
`%`	Modulus	`22 % 8`	6
`//`	Integer division	`22 // 8`	2
`/`	Division	`22 / 8`	2.75
`*`	Multiplication	`2 * 8`	16
`-`	Subtraction	`22 - 16`	6
`+`	Addition	`2 + 3`	5

Calculating rent

```
monthly_rent = 4000
yearly_rent = monthly_rent * 12
print(yearly_rent)
```

Money spent on coffee per yer

Okay, but how much do you spent for coffee-to-go per year?

Write a small Python program, which computes your yearly expenses for coffee-to-go.

Very likely it is important for your program that you know

- The price per cup
- How many cups of coffee you buy per week
- How many weeks you do that per year

In [ ]:

```
1
```

And how much would that be in Euro and US Dollars?

Extend your program, so that it computes your yearly expenses for coffee-to-go in Euro an USD. You can find daily exchange rates to DKK for example here: https://www.xe.com/currencyconverter/convert/?From=USD&To=DKK (https://www.xe.com/currencyconverter/convert/?From=USD&To=DKK)

In [ ]:

```
price_per_cup = 20
price_per_year = price_per_cup * 5 * 52
price_in_eur = price_per_year / 7.46465
price_in_usd = price_per_year / 6.61436

print(price_in_eur)
print(price_in_usd)
```

Integer, Floating-point, and String Data Types

Common data types

Data Type	Examples
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25
Strings	'a', 'aa', 'aaa', 'Hello!', '11 things'

Strings

A string is simply a series of characters. Anything inside quotes is considered a string in Python, and you can use single or double quotes around your strings.

In [ ]:

```
'Hello world!'
```

In [10]:

```
"Helloworld!"
```

Out[10]:

```
'Helloworld!'
```

In [11]:

```
"Hello world!" == 'Hello world!'
```

Out[11]:

```
True
```

String Concatenation and Replication

In [15]:

```
# Jeg bor i Kobenhavn
print('Hello' + 'World'+ '!')
```

```
HelloWorld!
```

In [16]:

```
'Hello' + 42
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-16-45471593d353> in <module>
----> 1 'Hello' + 42

TypeError: can only concatenate str (not "int") to str
```

In [17]:

```
'Hello' * 4
```

Out[17]:

```
'HelloHelloHelloHello'
```

In [ ]:

```
'Hello' * 'World'
```

The `None` Value

In Python there is a value called `None`, which represents the absence of a value. None is the only value of the `NoneType` data type. (Other programming languages might call this value `null`, `nil`, or `undefined`.) Just like the Boolean `True` and `False` values, `None` must be typed with a capital N. This value-without-a-value can be helpful when you need to store something that won't be confused for a real value in a variable. One place where `None` is used is as the return value of `print()`. The `print()` function displays text on the screen, but it does not need to return anything in the same way `len()` or `input()` does. But since all function calls need to evaluate to a return value, `print()` returns `None`.

In [ ]:

```
None
```

In [ ]:

```
result = print('Hello')
print(result)
```

Assignment Statements & Variables

A variable is initialized the first time a value is stored in it. When a variable is assigned a new value, the old value is *overwritten*.

In [ ]:

```
message = 'Hello'
message = 'World!'
print(message)
```

In [ ]:

```
message = 'Hello'
message = message + ' World!'
print(message)
```

In [18]:

```
message = 'Hello'
message += ' World!'
print(message)
```

```
Hello World!
```

Naming and Using Variables

When you're using variables in Python, you need to adhere to a few rules and guidelines. Breaking some of these rules will cause errors; other guidelines just help you write code that's easier to read and understand. Be sure to keep the following variable rules in mind:

Variable names can:

- contain only letters, numbers, and underscores
- start with a letter or an underscore
- not start with a number
- not contain spaces

In [ ]:

```
message_1 = 'Hello'
print(message_1)
```

In [19]:

```
1 1_message = 'Hello'
```

```
  File "<ipython-input-19-88f14f5bef67>", line 1
    1_message = 'Hello'
     ^
SyntaxError: invalid token
```

In [ ]:

```
1 greeting_message = 'Hej!'
```

Guidelines:

- Avoid using Python keywords and function names as variable names!
- Variable names should be short but descriptive.

For example, `name` is better than `n`, `student_name` is better than `s_n`, and `name_length` is better than `length_of_persons_name`.

Python keywords are the following:

`False`, `None`, `True`, `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `try`, `while`, `with`, `yield`

Python has 33 keywords.

So **never** do something like the following, it will just generate weird errors in your programs that are hard to find...

In [20]:

```
1 message = 'Important!'
2 print = 'Do not do this at home :)'
3 print(message)
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-20-815db8245b1a> in <module>
      1 message = 'Important!'
      2 print = 'Do not do this at home :)'
----> 3 print(message)

TypeError: 'str' object is not callable
```

Augmented Assignment Operators

When assigning a value to a variable, you will frequently use the variable itself.

```
>>> value = 42
>>> value = value + 1
>>> value
43
```

Instead, as a shortcut, you can use the augmented assignment operator `+=` to do the same.

There are augmented assignment operators for the `+`, `-`, `*`, `/`, and `%` operators.

Augmented Assignment Statement	Equivalent Assignment Statement
`value += 1`	`value = value + 1`
`value -= 1`	`value = value - 1`
`value *= 1`	`value = value * 1`
`value /= 1`	`value = value / 1`
`value %= 1`	`value = value % 1`

Note, the `+=` operator can also do string and list concatenation (adding something to the end of something else), and the `*=` operator can do string and list replication.

'#' Marks Comments

In [1]:

In [3]:

```
Hello World!
```

End-of-Line Terminates a Statement

The return key (↵) marks the end of a line.

In [4]:

```
Hello World!
Next Statement
```

So, programs really are just sequences of statements.

Expressions with Basic Operators

In [5]:

```
20
```

Nå, I thought the result will be `30` ...

Why isn't it?

In [6]:

Out[6]:

```
30
```

You can compute arbitrarily complex operations.

In [7]:

Out[7]:

```
16.0
```

But what happened here?

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Math operators from highest to lowest precedence:

Operator	Operation	Example	Evaluates to...
`**`	Exponent	`2 ** 3`	8
`%`	Modulus	`22 % 8`	6
`//`	Integer division	`22 // 8`	2
`/`	Division	`22 / 8`	2.75
`*`	Multiplication	`2 * 8`	16
`-`	Subtraction	`22 - 16`	6
`+`	Addition	`2 + 3`	5

Calculating rent

```
monthly_rent = 4000
yearly_rent = monthly_rent * 12
print(yearly_rent)
```

Money spent on coffee per yer

Okay, but how much do you spent for coffee-to-go per year?

Write a small Python program, which computes your yearly expenses for coffee-to-go.

Very likely it is important for your program that you know

- The price per cup
- How many cups of coffee you buy per week
- How many weeks you do that per year

In [ ]:

And how much would that be in Euro and US Dollars?

Extend your program, so that it computes your yearly expenses for coffee-to-go in Euro an USD. You can find daily exchange rates to DKK for example here: https://www.xe.com/currencyconverter/convert/?From=USD&To=DKK (https://www.xe.com/currencyconverter/convert/?From=USD&To=DKK)

In [ ]:

Integer, Floating-point, and String Data Types

Common data types

Data Type	Examples
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25
Strings	'a', 'aa', 'aaa', 'Hello!', '11 things'

Strings

A string is simply a series of characters. Anything inside quotes is considered a string in Python, and you can use single or double quotes around your strings.

In [ ]:

In [10]:

Out[10]:

```
'Helloworld!'
```

In [11]:

Out[11]:

```
True
```

String Concatenation and Replication

In [15]:

```
HelloWorld!
```

In [16]:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-16-45471593d353> in <module>
----> 1 'Hello' + 42

TypeError: can only concatenate str (not "int") to str
```

In [17]:

Out[17]:

```
'HelloHelloHelloHello'
```

In [ ]:

The `None` Value

In Python there is a value called `None`, which represents the absence of a value. None is the only value of the `NoneType` data type. (Other programming languages might call this value `null`, `nil`, or `undefined`.) Just like the Boolean `True` and `False` values, `None` must be typed with a capital N. This value-without-a-value can be helpful when you need to store something that won't be confused for a real value in a variable. One place where `None` is used is as the return value of `print()`. The `print()` function displays text on the screen, but it does not need to return anything in the same way `len()` or `input()` does. But since all function calls need to evaluate to a return value, `print()` returns `None`.

In [ ]:

In [ ]:

Assignment Statements & Variables

A variable is initialized the first time a value is stored in it. When a variable is assigned a new value, the old value is *overwritten*.

In [ ]:

In [ ]:

In [18]:

```
Hello World!
```

Naming and Using Variables

When you're using variables in Python, you need to adhere to a few rules and guidelines. Breaking some of these rules will cause errors; other guidelines just help you write code that's easier to read and understand. Be sure to keep the following variable rules in mind:

Variable names can:

- contain only letters, numbers, and underscores
- start with a letter or an underscore
- not start with a number
- not contain spaces

In [ ]:

In [19]:

```
  File "<ipython-input-19-88f14f5bef67>", line 1
    1_message = 'Hello'
     ^
SyntaxError: invalid token
```

In [ ]:

Guidelines:

- Avoid using Python keywords and function names as variable names!
- Variable names should be short but descriptive.

For example, `name` is better than `n`, `student_name` is better than `s_n`, and `name_length` is better than `length_of_persons_name`.

Python keywords are the following:

`False`, `None`, `True`, `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `try`, `while`, `with`, `yield`

Python has 33 keywords.

So **never** do something like the following, it will just generate weird errors in your programs that are hard to find...

In [20]:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-20-815db8245b1a> in <module>
      1 message = 'Important!'
      2 print = 'Do not do this at home :)'
----> 3 print(message)

TypeError: 'str' object is not callable
```

Augmented Assignment Operators

When assigning a value to a variable, you will frequently use the variable itself.

```
>>> value = 42
>>> value = value + 1
>>> value
43
```

Instead, as a shortcut, you can use the augmented assignment operator `+=` to do the same.

There are augmented assignment operators for the `+`, `-`, `*`, `/`, and `%` operators.

Augmented Assignment Statement	Equivalent Assignment Statement
`value += 1`	`value = value + 1`
`value -= 1`	`value = value - 1`
`value *= 1`	`value = value * 1`
`value /= 1`	`value = value / 1`
`value %= 1`	`value = value % 1`

Note, the `+=` operator can also do string and list concatenation (adding something to the end of something else), and the `*=` operator can do string and list replication.