Lesson 1: First steps with Python

Alessio Cardillo

Department of computer science and mathematics (DΣIM),
Universitat Rovira i Virgili, Tarragona, Spain

Python Days
Dept. of physics and astronomy – University of Catania, Catania, Italy

UNIVERSITAT ROVIRA i VIRGILI

Foreword

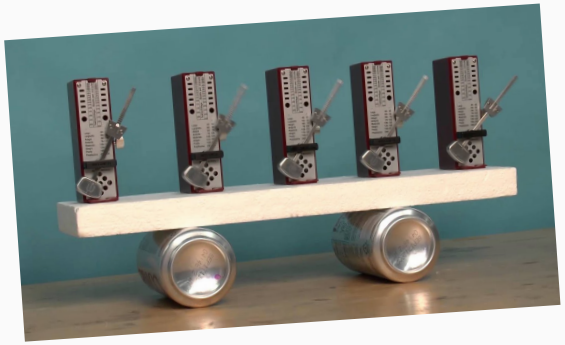
Who is Alessio Cardillo?

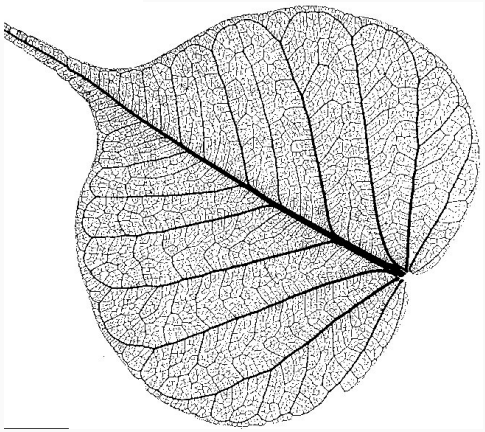
- MSc in physics at UniCT in 2010.

Who is Alessio Cardillo?

- MSc in physics at UniCT in 2010.
- PhD in physics at UniZar in 2014.
- More than 6 year of postdoctoral experience.

Who is Alessio Cardillo?





- MSc in physics at UniCT in 2010.
- PhD in physics at UniZar in 2014.
- More than 6 year of postdoctoral experience.
- Working on applications of statistical physics & nonlinear dynamics with a strong focus on numerical computation and data analysis.

Who is Alessio Cardillo?

- MSc in physics at UniCT in 2010.
- PhD in physics at UniZar in 2014.
- More than 6 year of postdoctoral experience.
- Working on applications of statistical physics & nonlinear dynamics with a strong focus on numerical computation and data analysis.
- Fortran 77 $\rightarrow$ C $\rightarrow$ Python

Outline of the course/lesson

What this course is

- An introduction (almost) from scratch to Python.
- A course oriented to people interested in scientific computing and quantitative data analysis.
- A primer on some mainstream applications (visualization and data analysis).
- The (personal) view of **a physicist** on the topic.

What this course is not

- A COMPLETE (and deep) course on the Python language.
- A tailored solution to single user's problems.
- A denigratory comparison between Python and other languages (*e.g.*, `C`).

Outline of the course/lesson

Course

1. Introduction to the basics of the Python language.

Lesson 1

- History & philosophy of Python
- Basics of Python
- (short) Hands-on session

Outline of the course/lesson

Course

1. Introduction to the basics of the Python language.
2. More “basics” of Python & basics of NumPy.

Outline of the course/lesson

Course

1. Introduction to the basics of the Python language.
2. More "basics" of Python & basics of NumPy.
3. IPython notebook & Visualization of data (Matplotlib).

Outline of the course/lesson

Course

1. Introduction to the basics of the Python language.
2. More "basics" of Python & basics of NumPy.
3. IPython notebook & Visualization of data (Matplotlib).
4. Data analysis (Pandas).

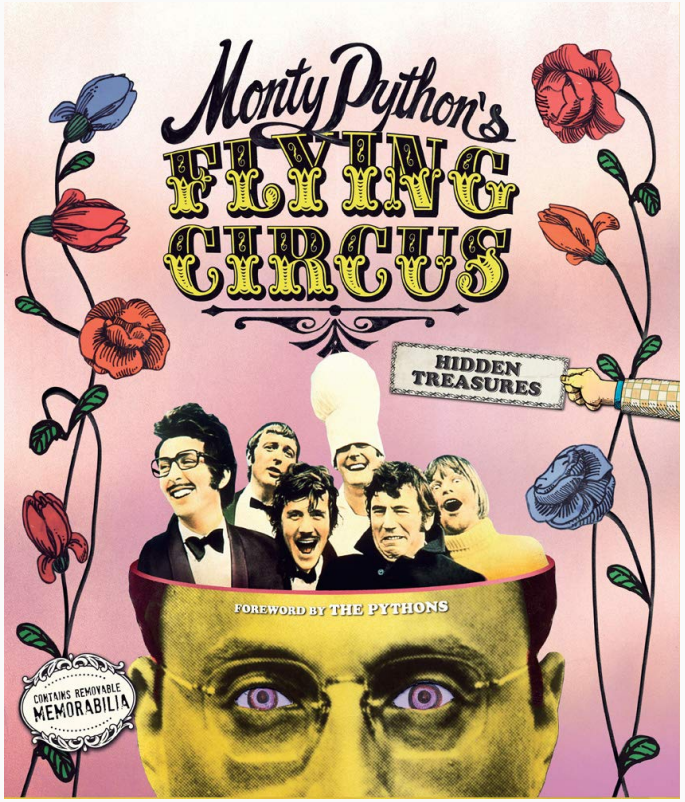
History & Philosophy of Python

Brief history & Zen of Python

- Developed in Amsterdam by a computer scientist named *Guido van Rossum*

Brief history & Zen of Python

- *"Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. [...] I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of* ***Monty Python's Flying Circus****)."*



Brief history & Zen of Python

- Up to 2018, van Rossum had the title of "*Benevolent Dictator For Life*" of Python

Brief history & Zen of Python

- Last release of Python2 is the 2.7.x (EOL Jan. 2020). Latest release of Python3 is 3.9.x

Brief history & Zen of Python

Zen of Python

The **Zen of Python** is a collection of 19 "*guiding principles*," written by Tim Peters, for writing computer programs that influence the design of the Python programming language.

Hands-on session

```
import this
```

Brief history & Zen of Python

Python vs Perl

Perl "*There's more than one way to do it*" (TMTOWTDI)

Python "*There should be one – and preferably only one – obvious way to do it.*"

Brief history & Zen of Python

Python vs Perl

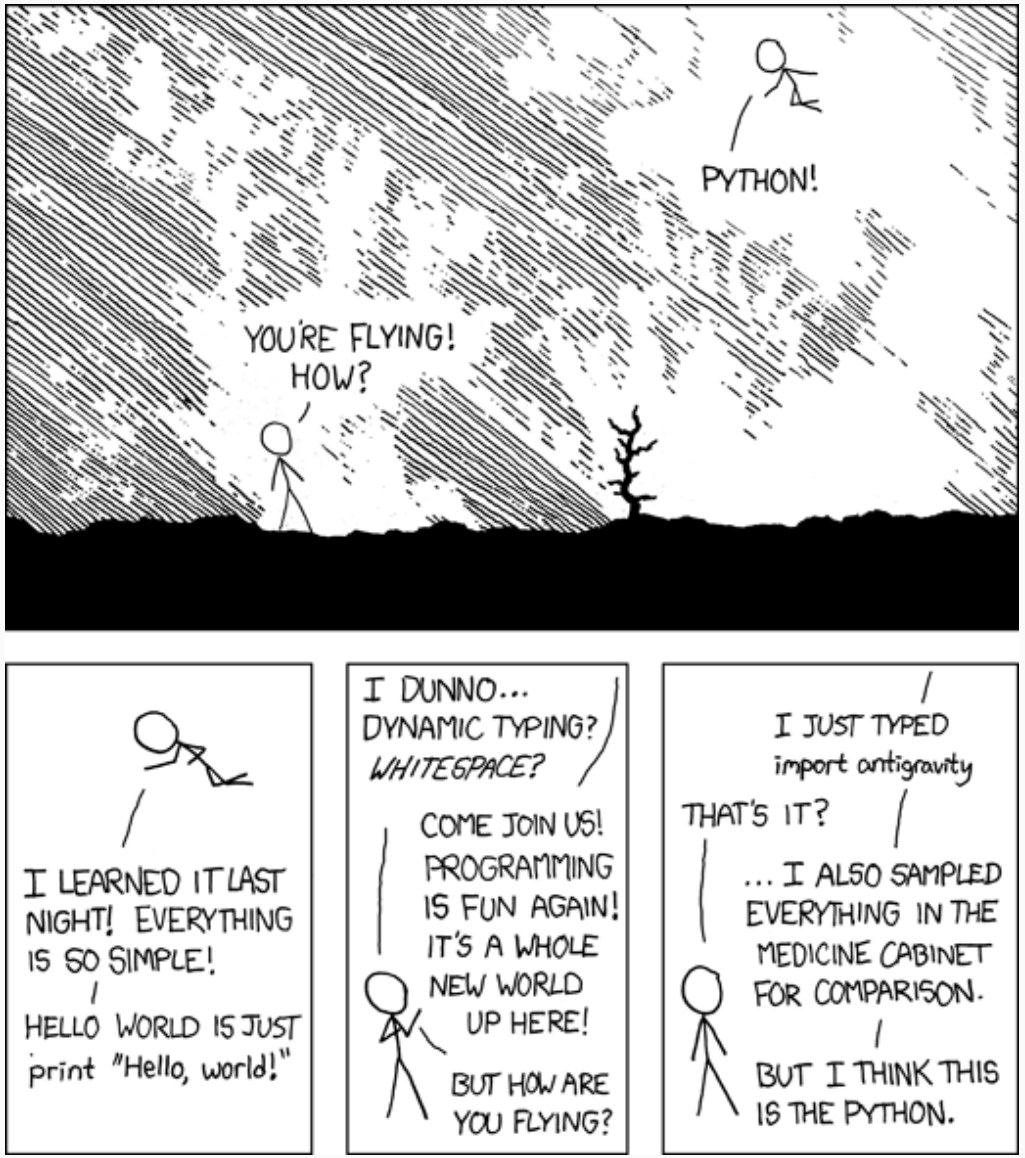
Perl "*There's more than one way to do it*" (TMTOWTDI)

Python "*There should be one – and preferably only one – obvious way to do it.*"

Perl Do we have a new "need"? Let's define a new operator!

Python The least amount operators and keywords we have, the better it is!

Why Python?



Python on XKCD. Available at https://xkcd.com/353/

Why Python?

Pros:

- It's **FREE** and available on many platforms/OS. Easy and **intuitive** syntax.
- **Multi-paradigm** (procedural, object oriented, functional)
- Ability to design/develop complex programs in a **fast way**.
- **Large availability** of libraries/modules.
- Huge community/diffusion (easy to get help).

Why Python?

Pros:

- It's **FREE** and available on many platforms/OS. Easy and **intuitive** syntax.
- **Multi-paradigm** (procedural, object oriented, functional)
- Ability to design/develop complex programs in a **fast way**.
- **Large availability** of libraries/modules.
- Huge community/diffusion (easy to get help).

Cons:

- It's an **interpreted** language (performances)
- Sometimes looks like a **black box** (debugging not always straightforward).

Differences between compiled and interpreted languages

Compiled

- To run the code you need to **compile** it (*i.e.*, convert into machine code) first, and then run the executable.
- The executable written in machine code is **tailored** on the "system" (architecture, OS, etc.) on which the source is compiled.
- Syntax (and others) errors are checked **during compilation**.

Differences between compiled and interpreted languages

Compiled

- To run the code you need to **compile** it (*i.e.*, convert into machine code) first, and then run the executable.
- The executable written in machine code is **tailored** on the "system" (architecture, OS, etc.) on which the source is compiled.
- Syntax (and others) errors are checked **during compilation**.

Interpreted

- The code is parsed by an **interpreter** which runs it into a virtual machine.
- The interpreter **translates** every instruction at runtime (affects performances).
- Syntax errors checks and debugging are done at **runtime** (*i.e.*, you may get errors only when the execution reaches a given instruction in the source code).

Code markup and PEP 8

- One of G. van Rossum's key insights is that "*code is read much more often than it is written*". Therefore: "**Readability counts**"!

PEP 8 – Style Guide for Python Code. Available at: `https://www.python.org/dev/peps/pep-0008/`

Code markup and PEP 8

- One of G. van Rossum's key insights is that "*code is read much more often than it is written*". Therefore: "**Readability counts**"!
- In Python the code's markup is not merely aesthetic: **it is part of the syntax**!

PEP 8 – Style Guide for Python Code. Available at: `https://www.python.org/dev/peps/pep-0008/`

Code markup and PEP 8

- One of G. van Rossum's key insights is that "*code is read much more often than it is written*". Therefore: "**Readability counts**"!
- In Python the code's markup is not merely aesthetic: **it is part of the syntax**!
- Python's markup makes the existence of delimiters like `;` `{ }` (C/C++), or statement like `END DO` (Fortran) superfluous.

PEP 8 – Style Guide for Python Code. Available at: `https://www.python.org/dev/peps/pep-0008/`

Code markup and PEP 8

Wrong

```
if␣(a␣>␣5):
print('a␣is␣greater␣than␣5')
else:
print('a␣is␣smaller␣than␣or␣equal␣to␣5')
```

Correct

```
if␣(a␣>␣5):
␣␣␣␣print('a␣is␣greater␣than␣5')
else:
␣␣␣␣print('a␣is␣smaller␣than␣or␣equal␣to␣5')
```

Code markup and PEP 8

Wrong

```
import sys, os
```

Correct

```
import sys
import os
```

Code markup and PEP 8

Wrong

```
foo = long_function_name(var_one, var_two,
      var_three, var_four)
```

Correct

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

Code markup and PEP 8

Wrong

```
x                = 1
y                = 2
long_variable = 3


spam( ham[ 1 ], { eggs: 2 } )
```

Correct

```
x = 1
y = 2
long_variable = 3


spam(ham[1], {eggs: 2})
```

Note:

These examples do not produce a Syntax Error but worsen the code's readability (Pet Peeves).

Basics of Python

The interpreter

- We can execute Python code either by writing it directly within the interpreter or by asking it to parse a script file.

Hands-on

`python`

or

`python myprogram.py`

Note

If you have multiple versions of Python installed on your machine, check to which one the command `python` points to!
(*e.g.*, `python3`)

The interpreter

- The interpreter compiles the source in bytecode and then the latter is executed by a virtual machine (*i.e.*, like Java).

The interpreter

- The interpreter is an excellent interactive calculator! It prints the value of every "meaningful" expression.

Hands-on

```
>>> 34 + 55 - 2**4
73
```

The interpreter

- The interpreter never dies, regardless of the severity of the error!

Hands-on

```
>>> 34 + 55/0 - 2**4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

Or

```
>>> 34 + 55 - log(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'log' is not defined
>>>
```

The interpreter

- The **MOST IMPORTANT** function in Python is `help()`!

Hands-on

```
>>> import numpy as np
>>> help(np.linspace)
```

- Stackoverflow: `https://stackoverflow.com/`

Our first program: `Hello, World!`

Our first program: `Hello, World!`

C

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!");
    return 0;
}
```

Python

```
>>> print('Hello, World!')
```

or equivalently

```
>>> print('Hello,', 'World!')
```

or even

```
>>> print('Hello,' 'World!')
```

Variables' type

In Python, you have the following built-in variable types:

Type	Denomination	Example
`int`	integer	`a = 5`
`float`	floating point	`a = 5.5`
`complex`	complex numbers	`a = 5 + 3j`
`bool`	boolean	`a = True`
`str`	string	`a = 'Hello'` `a = "Hello"`
`None`	None type	`a = None`

Hands-on

How do we know the type of a variable (*e.g.*, `x`)?

`type(x)`

Operators

Arithmetic

Operator	Denomination	Example
+	sum	`5 + 4 = 9`
-	subtraction	`5 - 4 = 1`
*	multiplication	`3 * 5 = 15`
/	division	`20 / 4 = 5`
%	modulus	`20 % 4 = 0`
**	power	`2**3 = 8`
//	floor division	`21 // 4 = 5`

Note

Python automatically recognizes the type of the variables and "adapts" the type of the result. For instance:

```
>>> 3 + 5
>>> 8
```

```
>>> 3 * 5.0
>>> 15.0
```

```
>>> 21 / 4
>>> 5.25
```

Operators

Assignment

Operator	Example	Equivalent to
=	`x = 5`	–
+=	`x += 5`	`x = x + 5`
-=	`x -= 5`	`x = x - 5`
*=	`x *= 5`	`x = x * 5`
/=	`x /= 5`	`x = x / 5`
%=	`x %= 5`	`x = x % 5`
**=	`x **= 5`	`x = x**5`
//=	`x //= 5`	`x = x//5`

Note

In agreement with the its philosophy, Python does not have the `++` and `--` operators (redundant).

Operators

Note

The = operator is much more than what it looks like. However, for our purposes, we can consider it a "copy" operator like in C or Fortran.

Hands-on

```
>>> a = 5
>>> b = 6
>>> print(a,b)
5 6
>>> a = b
>>> print(a,b)
6 6
>>> b = 4
>>> print(a,b)
6 4
```

Operators

Comparison

Operator	Denomination	Example
==	equal to	`x == y`
!=	not equal to	`x != y`
>	greater than	`x > y`
<	less than	`x < y`
>=	greater or equal to	`x >= y`
<=	less or equal to	`x <= y`

Logic

Operator	Description	Example
`and`	Returns True if both statements are True	`x > 5 and x < 10`
`or`	Returns True if one of the statements is True	`x < 5 or x < 10`
`not`	Reverse the result	`not(x > 5 and x < 10)`

Operators

Identity

Operator	Description	Example
`is`	Returns True if both variables are the same object	`x is y`
`is not`	Returns True if both variables are not the same object	`x is not y`

Membership

Operator	Description	Example

`in`	Returns True if a sequence with a given value is present in the object	`>>> a = [1,2,3,4]`
		`>>> 2 in a`
		`True`

`not in`	Returns True if a sequence is not present in the object	`7 not in a`

Data structures

Python comes with a **rich set** of built-in types of data structures!

`tuple`, `list`, `set`, and `dictionary`.

These structures can be combined together and allow for nested levels. Such an availability of data structures (one of the main advantages of Python) makes the code more versatile, allowing to tackle more easily complex problems. In other languages (*e.g.*, C or Fortran) either you have to "import" these data structures using external libraries, or you have to implement them from scratch.

- https://docs.python.org/3/tutorial/datastructures.html

Tuples

A **tuple** is a container of objects that **cannot be modified**!

Hands-on

```
>>> a = tuple() # empty tuple
>>> b = () # empty tuple
>>> print(a)
()
>>> type(b)
<class 'tuple'>

>>> a = 1,2,3  # filled tuple
>>> print(a)
(1, 2, 3)
>>> type(a)
<class 'tuple'>

>>> b = (4,5,6) # filled tuple
>>> print(b)
(4, 5, 6)
```

```
>>> a = (1,2,3) # filled tuple
>>> a[0] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support
item assignment
```

Tuples

Note1

The elements of a tuple do not have to be of the same type! (To have the same in C, you need to use a `struct`)

```
>>> a = (1,'ciao!',-125.4)
>>> type(a)
<class 'tuple'>
```

Note2

Tuples can be nested!

```
>>> c = (1,2,("alpha","beta"), 12.4, -3.5)
>>> type(c)
<class 'tuple'>
>>> type(c[0])
<class 'int'>
>>> type(c[2])
<class 'tuple'>
```

Tuples

Question

How many elements has a tuple? The answer is provided by the `len()` function!

We can perform **operations** with tuples.

Hands-on

```
>>> a = (1,'ciao!',-125.4)
>>> len(a)
3
```

```
>>> a = (1,2,3)
>>> b = (4,5,6)
>>> a+b
(1, 2, 3, 4, 5, 6)
>>> a*2
(1, 2, 3, 1, 2, 3)
>>> a*b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence
by non-int of type 'tuple'
```

Lists

A **list** is a "*modifiable*" tuple.

Hands-on

```
>>> a = list()  # empty list
>>> b = [] # empty list
>>> type(a)
<class 'list'>
>>> type(b)
<class 'list'>
>>> print(a)
[]
>>> b = [4,5,6] # filled list
>>> print(b)
[4, 5, 6]
```

```
>>> a = [1,2,3]
>>> print(a)
[1, 2, 3]
>>> a[0] = -1
>>> print(a)
[-1, 2, 3]
```

Lists

Lists are more "powerful" than tuples ...

Hands-on

```
>>> a = [1,10,34,-5,3.9,'hello']
>>> a.append('world')  # appending a value to a list
>>> print(a)
[1, 10, 34, -5, 3.9, 'hello', 'world']
>>> a.remove(34)  # removing a value from a list
>>> print(a)
[1, 10, -5, 3.9, 'hello', 'world']
>>> a.extend([-1,-1]) # extending a list (alternatively, we can sum them)
>>> print(a)
[1, 10, -5, 3.9, 'hello', 'world', -1, -1]
>>> a.insert(2, -100.) # inserting (-100) at a given position (2)
>>> print(a)
[1, 10, -100.0, -5, 3.9, 'hello', 'world', -1, -1]
>>> b = a.pop(5) # extracting the 5-th element of the list
>>> print(a, "\n", b)
[1, 10, -100.0, -5, 3.9, 'world', -1, -1]
hello
```

Lists

Hands-on

```
>>> a = [1, 3, 5, 5, 7, 9, 1]
>>> a.index(5) # return the index of the first occurrence of (5)
2
>>> a.count(1) # counts how many occurrences of (1) there are
2
>>> a.sort() # sort the list
>>> print(a)
[1, 1, 3, 5, 5, 7, 9]
>>> a.reverse() # reverse the elements of a list
>>> print(a)
[9, 7, 5, 5, 3, 1, 1]
```

List & membership

```
>>> a = [1, 3, 5, 5, 7, 9, 1]
>>> 7 in a
True
>>> -7 in a
False
```

We can use the membership operators to verify if a certain "object" belongs to a data structure

Sets

A **set** is an ensemble of unique elements. Sets are not "ordered" (*i.e.*, they are ordered automatically to speed up search operations)

Hands-on

```
>>> a = set() # empty set
>>> type(a)
<class 'set'>
>>> print(a)
set()
>>> len(a) # set's number of elements
0
>>> a.add(1) # adding an element to the set
>>> a
{1}
>>> len(a)
1
```

Sets

Hands-on

```
>>> len('abracadabra')
11
>>> a = set('abracadabra')
>>> print(a)
{'b', 'd', 'c', 'r', 'a'} # the order is changed!
>>> len(a)
5
```

Sets

On sets, we can perform operations like **union**, **intersection**, and **difference** (also symmetric).

Note

There exists also the "function" version of set operators (*e.g.*, `A.union(B)` is equivalent to `A | B`).

Hands-on

```
# sets are defined
A = {0, 2, 4, 6, 8}
B = {1, 2, 3, 4, 5}

# union
print("Union :", A | B)

# intersection
print("Intersection :", A & B)

# difference
print("Difference :", A - B)

# symmetric difference
print("Symmetric difference :", A ^ B)
```

Sets

Note

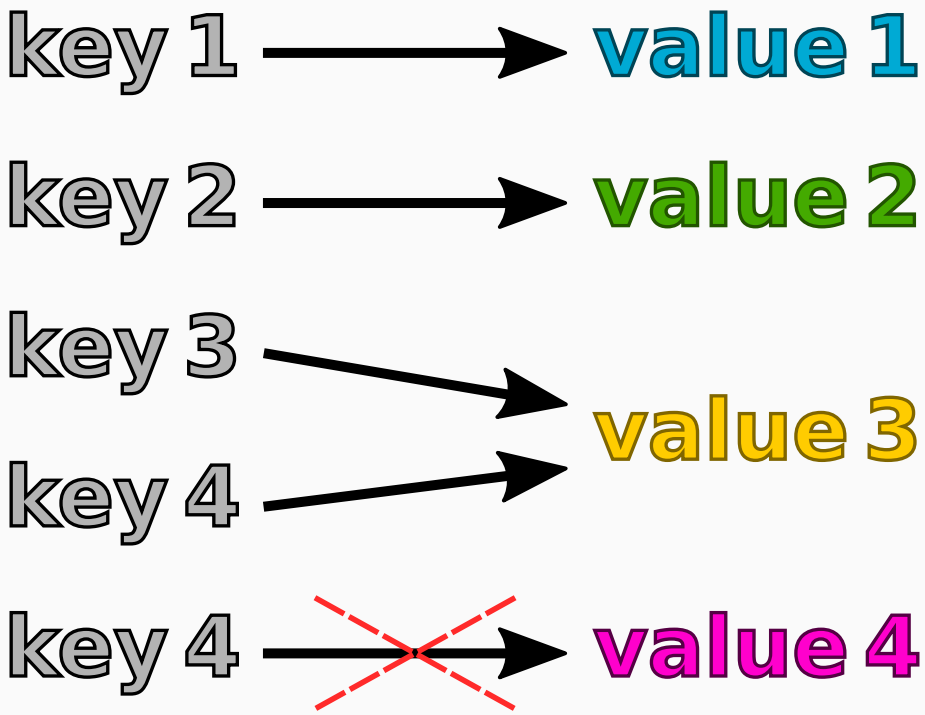
The equivalent of tuples for sets are called **frozenset**.

Hands-on

```
>>> b = frozenset([0,1,5])
>>> type(b)
<class 'frozenset'>
>>> print(b, '  Nr. elements = ', len(b))
frozenset({0, 1, 5})   Nr. elements =  3
>>> b.add(7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

Dictionaries

Dictionaries are an implementation of associative arrays. They are structures implementing a correspondence between a key and a value. Like sets, dictionaries are not ordered, and their "frozen version" is called frozendict.



- https://en.wikipedia.org/wiki/Associative_array

Dictionaries

Hands-on

```
>>> a = dict() # empty dictionary
>>> b = {} # empty dictionary
>>> type(a)
<class 'dict'>
>>> type(b)
<class 'dict'>
>>> # you can initialize a dict by filling it!
>>> a = {'key1': (10,20), 'key2': None}
>>> print(a)
{'key1': (10, 20), 'key2': None}
>>> a['key3'] = {'key31': 'mystr'} # adding a new key
>>> print(a)
{'key3': {'key31': 'mystr'}, 'key1': (10, 20), 'key2': None}
```

Playing with dictionaries . . . I

As dictionaries are "*not ordered*" containers, it is important to understand how to use them.

Hands-on

```
>>> a = {'key1': (10, 20), 'key2': None, 33: {'key31': 'mystr'}}
>>> # listing the dictionary keys
>>> a.keys()
dict_keys([33, 'key1', 'key2'])
>>> # listing the dictionary values
>>> a.values()
dict_values([{'key31': 'mystr'}, (10, 20), None])
>>> # accessing the value corresponding to a given key
>>> print(a['key1'])
(10, 20)
```

Dictionaries

Playing with dictionaries ... II

How can we "visualize" a dictionary? We can use the method `.items()`!

Hands-on

```
>>> for k,v in a.items():
...     print('key = ', k, '\tvalue = ', v)
...
key =  33          value =  {'key31': 'mystr'}
key =  key1          value =  (10, 20)
key =  key2          value =  None
```

Handy trick

To verify if a dictionary has a certain key, we can use:

```
>>> key in dict
```

Strings

A **string** is a *tuple of characters*.

Hands-on

```
>>> a = 'abracadabra'
>>> len(a)
11
>>> a[1]
'b'
>>> a[1] = 'c'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Strings

A string's delimiter could be ' or ". We can use either one or three consecutive delimiters to define a string. This becomes handy when quotes are part of the string itself.

Example

```
>>> a = 'aaa'
>>> b = '''aaa'''
>>> a == b
True
>>> c = '''this is a veeery long string spanning two
... lines having both 'single' and "double" quotes'''
>>> type(c)
<class 'str'>
```

Strings

Playing with strings ...

Like for tuples, we can perform operations on strings

```
>>> a = 'abra'
>>> b = 'cadabra'
>>> print(a+b)
abracadabra
>>> print(a+'\t'+b)
abra        cadabra
```

Note

Python uses the same escape characters as `C` (*i.e.*, `\n`, `\t`, and `\r`).

Strings

Playing with strings ...

```
>>> a = 'Hello World!'
>>> print(a.lower())
hello world!
>>> print(a.upper())
HELLO WORLD!
>>> print(a.replace('o','x')) # replacing 'o' with 'x'
Hellx Wxrld!
>>> print(a.find('lo')) # returns the position of 'lo'
3
>>> print(a.split(' ')) # split when a space ' ' occurs
['Hello', 'World!']  # Note: the output is a list!
```

Strings

Playing with strings ...

```
>>> a = 'Hello World!'
>>> print(a.tofrench())
Bonjour tout le monde!
>>> print(a.tospanish())
Hola a todo el mundo!
>>> print(a.toitalian())
Ciao mondo!
```

Strings

Playing with strings ...

```
>>> a = 'Hello World!'
>>> print(a.tofrench())
Bonjour tout le monde!
>>> print(a.tospanish())
Hola a todo el mundo!
>>> print(a.toitalian())
Ciao mondo!
```

Just kidding, these functions do not exist! :-P

Control of flow

In accordance with its philosophy, Python does not have many statements to control the flux of the instructions. There are, basically, three statements:

`if-elif-else`, `while`, and `for`

- https://docs.python.org/3/tutorial/controlflow.html

The `if` statement

basic `if`

```
>>> if a == b:
...     print('a is equal to b')
... else:
...     print('a is not equal to b')
...
```

"advanced" `if`

```
>>> if a < 5:
...     print('a is less than five')
... elif a == 5:
...     print('a is equal to five')
... else:
...     print('a is more than five')
...
```

Note 1

We can combine conditions (*e.g.*, `((a >= 5) and (a <= 10))`)

Note 2

Contrary to `C`, Python does not have a `switch` statement.

(less operators = better!)

The `while` loop

The `while` statement is a loop with a condition

```
>>> i = 0
>>> mycheck = True
>>> while mycheck: # runs until True
...     print('i = ', i)
...     i += 1
...     if i > 5:
...         mycheck = False
...
i =  0
i =  1
i =  2
i =  3
i =  4
i =  5
```

Note

If not handled carefully, `while` loops can become **endless**!

The for loop

C

```c
#include <stdio.h>

int i;

int main()
{
    for(i=0; i<10; i++)
    {
        printf("i = %d\n", i);
    }

    return 0;
}
```

Python

```
>>> for i in range(10):
...     print('i = ', i)
...
```

The `for` loop

The `range` function

The function `range` returns a so-called *iterator*.

```
range(start, stop, step)
```

It produces a sequence of integers from `start` (included) to `stop` (excluded) spaced of `step`.

Hands-on

```
>>> print([i for i in range(5)])
[0, 1, 2, 3, 4]
>>> print([i for i in range(0,15,3)])
[0, 3, 6, 9, 12]
>>> print([i for i in range(100,0,-20)])
[100, 80, 60, 40, 20]
```

The `for` loop

Note

In reality, the `for` statement in Python is much more "powerful" than its `C` counterpart!

In fact, the `for` statement allows to iterate over any *iterable* object.

The for loop

C

```c
#include <stdio.h>

int i;
int vals[] = {10,15,20,25};

int main()
{
   for(i=0; i<4; i++)
   {
      /* do something with array */
      printf("val[%d] = %d\n", i, vals[i]);
   }

   return 0;
}
```

Python

```
>>> for i in [10,15,20,25]:
...     print('i = ', i)
...
```

alternatively

```
>>> a = [10,15,20,25]
>>> for i in range(len(a)):
...     print('val[',i,'] = ', a[i])
...
```

The `for` loop

Hands-on

```
>>> a = {'key1': 10,
...      'key2': [2., 3.5],
...      3: {'key31': None, 'key32': ('alpha', 'beta')}
...     }
>>> for k in a.keys():
...     print(k)
...
3
key1
key2
```

The break & continue

break

It forces the exit from the cycle

Hands-on

```
>>> for i in range(10000):
...     print('i = ', i)
...     if i%3 == 2:
...         break
...
i =  0
i =  1
i =  2
```

continue

Allows to "jump" to the next iteration

Hands-on

```
>>> for i in range(4):
...     if i == 1:
...         continue
...     print('i = ', i)
...
i =  0
i =  2
i =  3
```

The print function

In Python 3, the print function has become much more powerfull/flexible than before (in Python 2.7 it was a statement)!

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Examples

```
>>> a = ['This', 'is', 'a', 'sentence']
>>> print(*a)  # the *a "expands" the content of a
This is a sentence
>>> # we can change the separator between the elements of a
>>> print(*a, sep=' *.* ')
This *.* is *.* a *.* sentence
>>> # we can define what is appended at end of the print
>>> print(*a, sep=' *.* ', end='\nEND\nCIAO CIAO!\n')
This *.* is *.* a *.* sentence
END
CIAO CIAO!
```


- https://docs.python.org/3/library/functions.html#print

The `print` function

Question

How can we **format** the output of the `print`?

The print function

Examples

```
>>> val1 = 35.866
>>> val2 = 22.0
>>> # formatting a string
>>> s = 'val1 = {:.2f} while val2 = {:.0f}'.format(val1, val2)
>>> print(s)
>>>
>>> # using str format directly inside print
>>> print('val1 = {:.2f} while val2 = {:.0f}'.format(val1, val2))
>>>
>>> # using string literals (or f-strings) (only for Python > 3.6.x)
>>> print(f'val1 = {val1:.2f} while val2 = {val2:1}')
>>>
>>> # The "old" way
>>> print('val1 = %.2f while val2 = %d' % (val1, val2))
```

They all produce the same output!

$$\text{val1} = 35.87 \text{ while } \text{val2} = 22$$

- https://docs.python.org/3/tutorial/inputoutput.html

Hands-on Session

Exercise 1

Tasks

1. Compute the product between the following matrices

$$\mathcal{A} = \begin{pmatrix} 0 & 1 & -5 \\ 3 & 10 & 2 \\ -1 & 1 & 0 \end{pmatrix} \quad \mathcal{B} = \begin{pmatrix} 7 & 3 & -4 \\ 1 & 0 & -1 \\ 5 & 2 & -5 \end{pmatrix},$$

   and print it on screen.

2. Generate lists of 5, 10, 20, 50, 100, 1000, 50000, 1000000, and 10000000 random numbers in the $[0, 1)$ interval, compute their averages and standard deviations, and print their evolution with respect to the size of the sample (with a precision of 5 and 8 decimals, respectively).

Exercise 1

Random numbers

To generate random numbers in the $[0, 1)$ interval, we can use the following code

```
# load the random module
# (do only once)
import random as rnd

# extract a random number
r = rnd.random()
```

Square root

The square root function is available through the `math` module

```
import math

# compute the square root
mysqrt = math.sqrt()
```

Tip

There are ways to compute the standard deviation without the need to store all the values. See: `https://en.wikipedia.org/wiki/Standard_deviation#Rapid_calculation_methods`

Exercise 2

Tasks

After loading Martin Scorsese's filmography from file, arrange the information into a dictionary and answer to the following questions:

1. How many movies has he directed?
2. What is the time span of Scorsese's career?
3. What is the title of his oldest movie?
4. How many movies he has also produced?
5. How many movies he has produced and written?

Print the answers on screen all at once!

Exercise 2

Load the input file

```
dirin = '<relative path to the file>'
fnamein = 'scorsese_filmography.txt'

fnamein = dirin+fnamein

with open(fnamein) as fin:
    next(fin) # skipping the first row
    for line in fin:
        # splitting each line into
        # a list of strings
        mystr = line.split(";")

        # Transfer the content of mystr
        # into variables
```

maximum & minimum

```
a = [1,2,3]


mymax = max(a)
mymin = min(a)
```

Caution

The lines of the file are parsed as `strings`! You have to convert them (*e.g.*, using the `int()` function).

Bibliography i

- Homepage of the Python programming language. Available at: `https://www.python.org/`
- HTML.it, *Guida al linguaggio Python*. Available at: `https://www.html.it/guide/guida-python/`
- Python PEP-8 directive. Available at: `https://www.python.org/dev/peps/pep-0008/`
- Homepage of the Stackoverflow website. Available at: `https://stackoverflow.com/`
- Python 3 documentation. Available at: `https://docs.python.org/3/`

Bibliography ii

- Python 3 documentation, *Data structures*. Available at: `https://docs.python.org/3/tutorial/datastructures.html`
- Associative array. Available at: `https://en.wikipedia.org/wiki/Associative_array`
- Python 3 documentation, *Control of flow*. Available at: `https://docs.python.org/3/tutorial/controlflow.html`
- Python 3 documentation, *The* `print` *function*. Available at: `https://docs.python.org/3/library/functions.html#print`
- Python 3 documentation, *Output format*. Available at: `https://docs.python.org/3/tutorial/inputoutput.html`

Bibliography iii

- Wikipedia, *Rapid calculation method for standard deviation*. Available at: https://en.wikipedia.org/wiki/Standard_deviation#Rapid_calculation_methods
- Martin Scorsese's filmography. Available at: https://en.wikipedia.org/wiki/Martin_Scorsese_filmography