SCRIPTING AND PROGRAMMING LABORATORY FOR DATA ANALYSIS

Lecture 1

Introduction to python and basic programming tools

STUDENT SURVEY

Go to: https://www.wooclap.com/SCRIPTPY

CODE SCRIPTPY

1. What is your programming experience?
2. Which language do you already know?
3. Which operative system are you familiar with?
4. What is your current knowledge of python?

PROGRAM OF THE COURSE

1. Introduction to python
2. Conditional/recursive statements
3. Simple and structured data types
4. Modules and functions, introspection
5. Files and directories management
6. Numpy
7. Scipy
8. Matplotlib

Hands on examples:

9. Solving ODEs: two body problem
10. Fast Fourier transform
11. Bot programming?
12. Machine learning
13. Advanced: pandas, object programming, language interfaces

FINAL PROJECT AND FINAL EXAM

The final exam will consist of a project you will carry out based on what you will learn during the course.

You will be asked to select a set of data (provided by us/proposed by you/expanding what has been done during the lectures) and analyze it extracting useful information from the data.

You will present the results in a jupyter notebook

A basic introduction to python

About coding

Coding is telling a computer to do something using a language it understands. The instructions must be **non ambiguous** and the **construct has to be logical**. The ordered set of unambiguous instructions is generally called ***algorithm***.

There are many different programming languages with different features. Languages are human readable and human writable instructions that can be converted into machine readable instructions via a “translator” (the compiler).

The current course will teach the **python** programming language.

WHAT IS PYTHON?

*"Python is an **interpreted**, **object-oriented**, **high-level** **programming language with dynamic semantics**. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for* Rapid Application Development, *as well as for use as a* scripting *or* glue language *to connect existing components together. Python's simple, easy to learn syntax **emphasizes readability** and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be **freely distributed**."*

Source: https://www.python.org/doc/essays/blurb/

In short detail

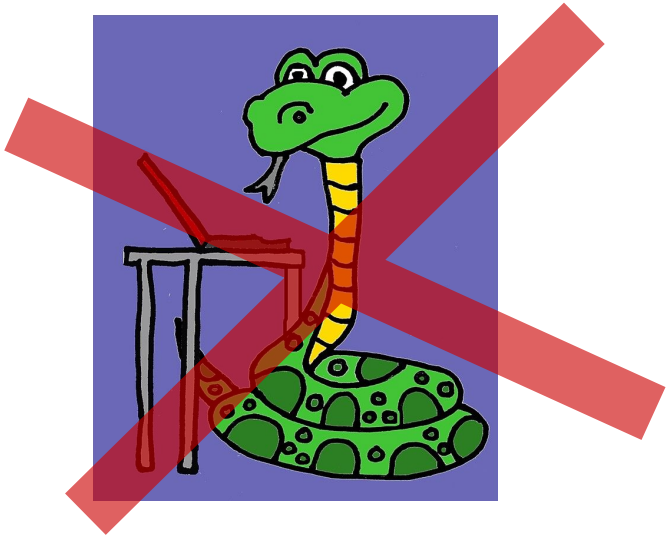
Python properties:

- high level
- interpreted
- dynamic semantics
- rapid application development
- easy to read
- can be used for scripting
- modular
- *(object oriented)*
- *(glue language)*

The history of Python

Python was conceived in the late 1980s by **Guido van Rossum** in the Netherlands as a descendant of the “ABC language”.

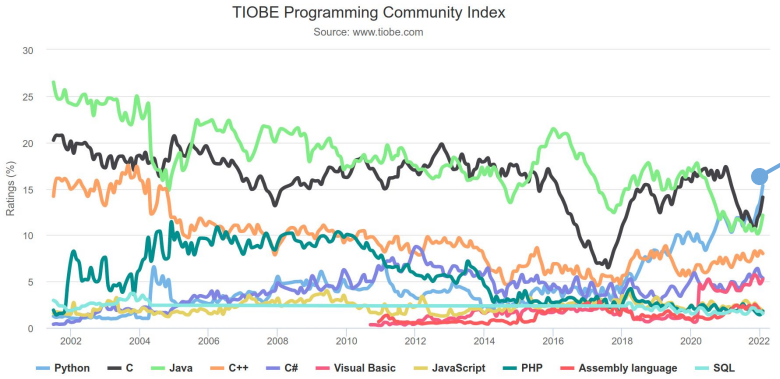
Named after…



The history of Python

- Python 1: first prototype realized in 1991-1994; in 1999 van Rossum created an initiative called *Computer Programming for Everybody* (CP4E) to make programming more accessible
- Python 2: released by the “BeOpen python Lab” in 2000; has a more community-oriented approach. Was supported till 2020.
- Python 3: released in 2008, with some flaws fixes; it is the version to be used nowadays!
- **Python is entirely open source, in all its releases!**

Python today



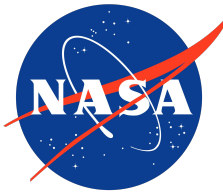
https://www.tiobe.com/tiobe-index

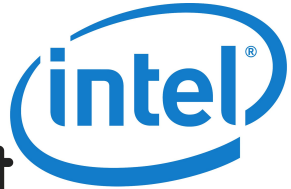
Main usage of python:
Data analysis, machine learning, data visualization, web development, scripting…

Feb 2022	Feb 2021	Change	Programming Language	Ratings	Change
1	3	↑	Python	15.33%	+4.47%
2	1	↓	C	14.08%	-2.26%
3	2	↓	Java	12.13%	+0.84%
4	4		C++	8.01%	+1.13%
5	5		C#	5.37%	+0.93%
6	6		Visual Basic	5.23%	+0.90%
7	7		JavaScript	1.83%	-0.45%
8	8		PHP	1.79%	+0.04%
9	10	↑	Assembly language	1.60%	-0.06%
10	9	↓	SQL	1.55%	-0.18%
11	13	↑	Go	1.23%	-0.05%
12	15	↑	Swift	1.18%	+0.04%
13	11	↓	R	1.11%	-0.45%
14	16	↑	MATLAB	1.03%	-0.03%
15	17	↑	Delphi/Object Pascal	0.90%	-0.12%
16	14	↓	Ruby	0.89%	-0.35%
17	18	↑	Classic Visual Basic	0.83%	-0.18%

The (bright) future of Python

Python success stories





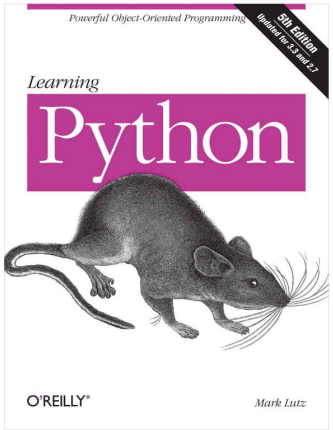
Books and documentation

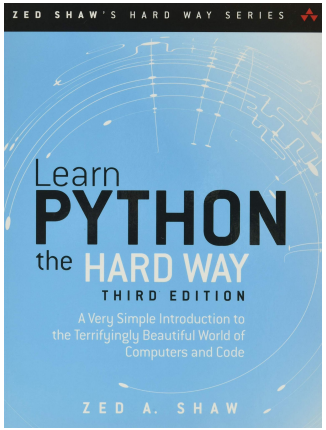
Official documentation:

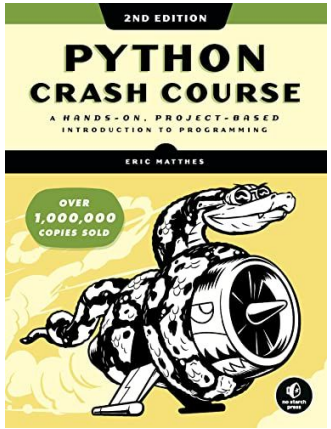
https://docs.python.org/3/

https://www.python.org/doc/

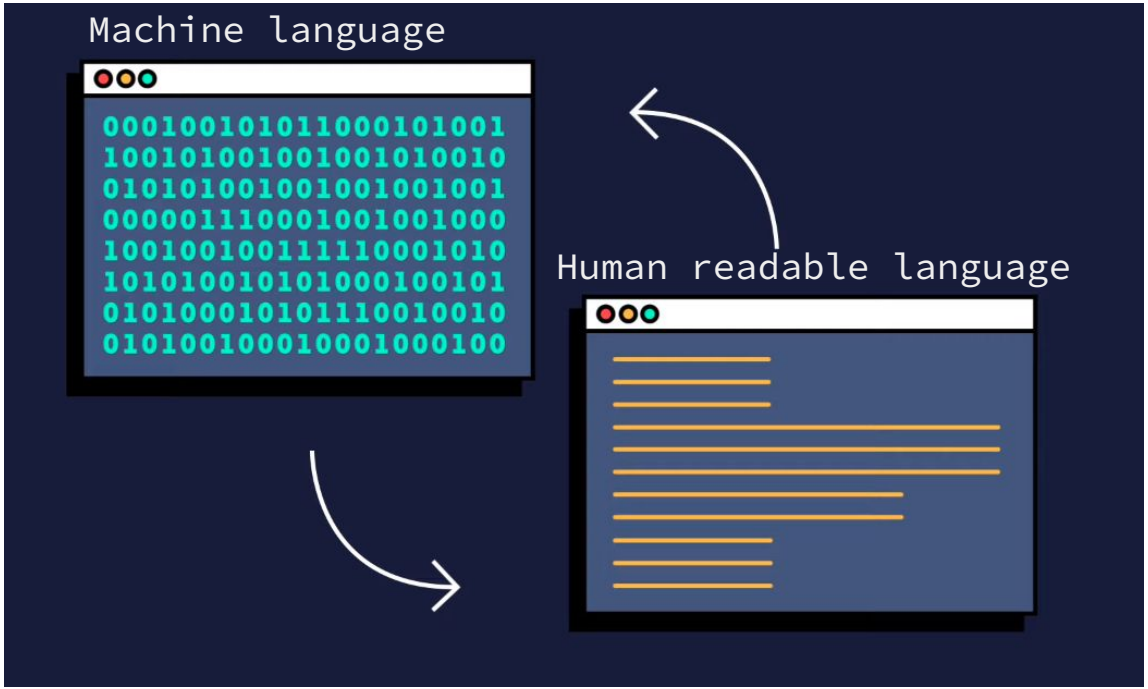
Books suggestions:







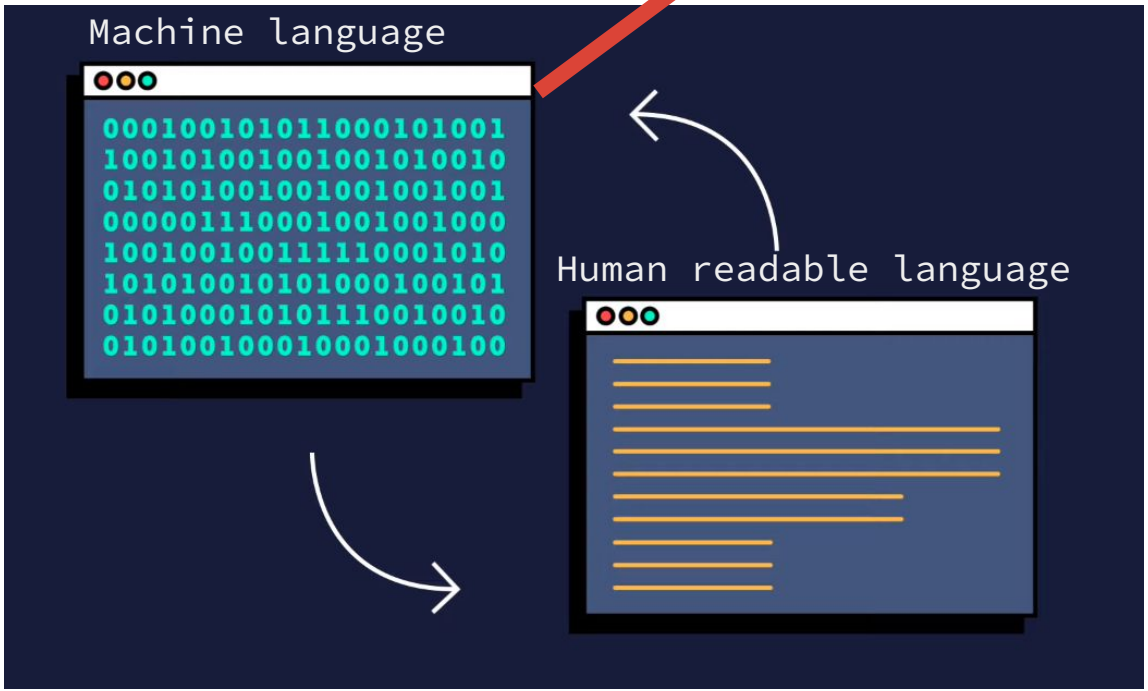
Properties of Programming Languages



Binary

https://www.wooclap.com/SCRIPTPY

CODE SCRIPTPY



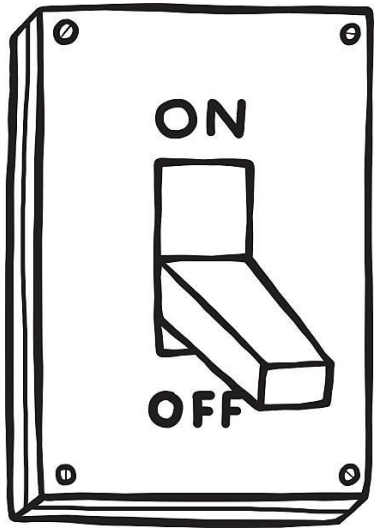
A very short introduction to binary

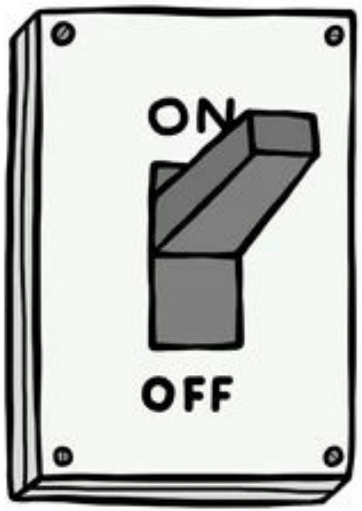
Used by computers is a number system that uses only

0 1

A very short introduction to binary

Used by computers is a number system that uses only

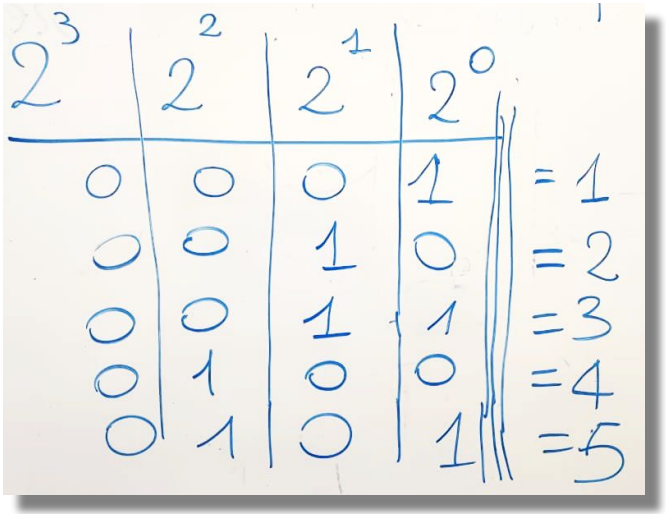




A VERY SHORT INTRODUCTION TO BINARY

Used by computers is a number system that uses 0s and 1s.

Integers can be represented in base 2 (binary) in this way, so that:



Convert 1011 from binary to decimal base

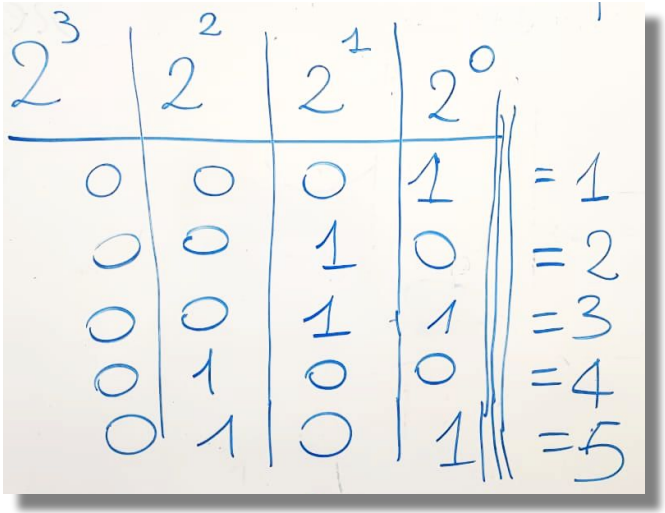
https://www.wooclap.com/SCRIPTPY

CODE SCRIPTPY

A VERY SHORT INTRODUCTION TO BINARY

Used by computers is a number system that uses **0s** and **1s**.

Integers can be represented in base 2 (**binary**) in this way, so that:



Each 0 or 1 in the memory of a computer is called **bit**. A sequence of 8 bits is the famous **byte**.

In python2, an integer used to occupy 32 bits=8 bytes. The first bit was used to set the sign of the number (0, or 1). [Note: python3 does not have this limitation anymore.]

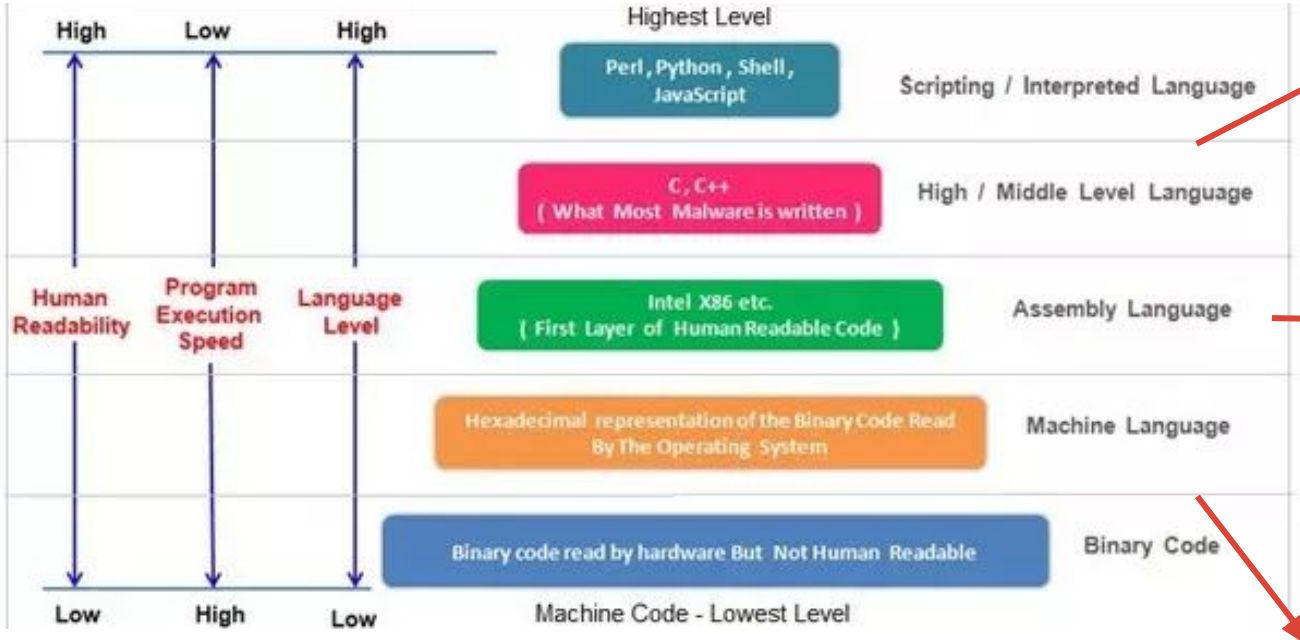
What was the maximum integer that could be stored?

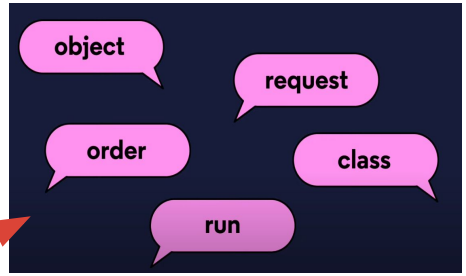
A very short introduction to binary

This approach is also used to represent characters that can make up strings and floating point numbers (mantissa, exponent, sign).

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111

Low vs High level programming languages



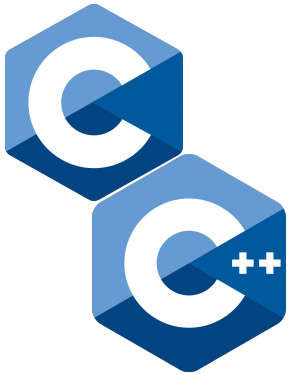


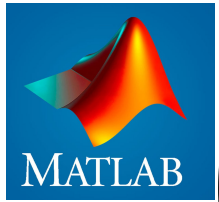

```
cseg segment
        assume cs:cseg,
ds:cseg, ss:cseg
        org 100h
        .386
start:

        mov ax, 13h
        int 10h

        mov dx, 3c8h
        xor al, al
        out dx, al
        inc dx
```

```
00010010101100010100
10010100100100101010
01010100100100100100
00000111000100100100
10010010011111000101
10101001010100010010
01010001010111001001
01010100010001000100
```

Or: how close one is to the machine, what is the level of abstraction when writing the instructions for the computer.

Languages: compiled vs *interpreted* (or better: bytecode pre-compiled)





Perl

Source code → Compile the code and get the executable → Run the executable

```
int main(void)
{
  for (int i = 0; i<5; i++)
    cout<< i<< endl;
  return 0;
}
```

```
>> gcc -c prog.cpp
>> gcc -o prog prog.o
```

```
>> ./prog
```

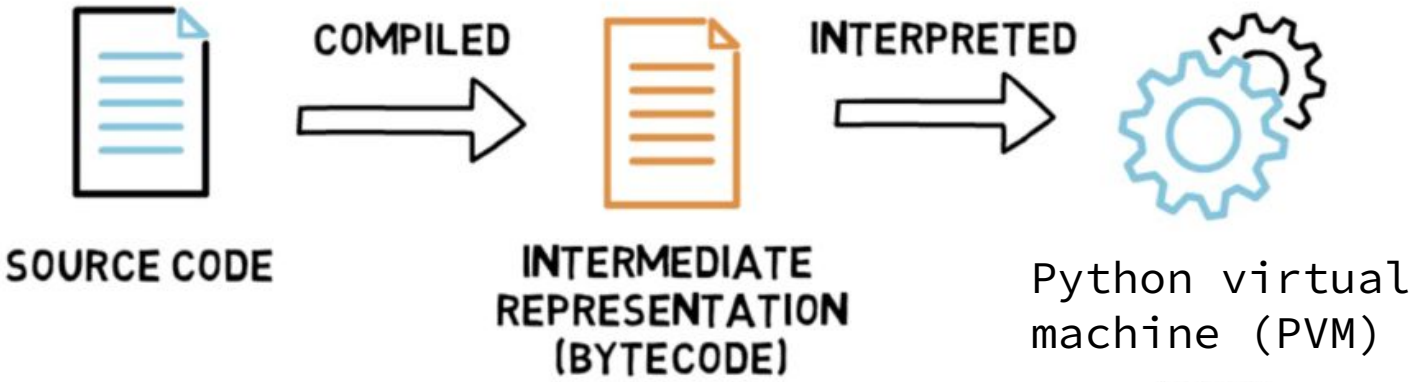
Source code → Interpreter acts on the fly and the source code can be “executed” immediately

```
for i in range(5):
  print(i)
```

```
>> python prog.py
```

https://www.wooclap.com/SCRIPTPY

Nowadays: Bytecode intermediate step for interpreted lang.



1. Python compiler reads a source code (**.py**) or instruction. It checks the syntax of each line. If it encounters an error, it immediately shows an error.
2. The compiler translates the source code into BYTECODE, its equivalent form in an intermediate language (**.pyc, .pyo files**)
3. BYTECODE is sent to the interpreter, the Python Virtual Machine (PVM). PVM converts the BYTECODE into a machine-executable code. If an error occurs during this interpretation then the conversion is halted with an error message.

⇒ Python does not require a “separate” compilation, and its hybrid approach makes it relatively efficient to be interpreted

Costs versus benefits

- **Execution speed:** compiled languages typically feature better performances
- **Development speed:** interpreted languages are generally higher level languages, the code development is simpler and faster: the syntax is easier, it takes less time and lines to write a given command; errors can be spotted faster, skipping the manual compilation step
- **Code portability:** how easy and immediate it is to use a given code in a platform different than the one in which it was developed:
  - Compiled language: the executable is portable only on platforms with properties analogues to the original one; the source code can be re-compiled in the new platform, but the needed compilers and libraries need to be present in the new machine!
  - Interpreted language: the only prerequisite is the existence of a proper interpreter in the new platform

Typing: statically vs dynamically typed

- **Static typing:** adopted mainly by compiled languages as C/C++, Fortran. The type of variable is decided in the moment the variable is defined, and cannot be changed within the scope in which it is defined.

Example (C):

```c
void prova(void){
  float a=5.0;
  return;
}

int main(void){
  int a=0;
  int b=5;
  b=a;
  return 0;
}
```

- **Dynamic typing:** most of interpreted languages have this feature. There is no need to decide in advance the type of a variable, which is inferred depending on its assignment. The same variable can be assigned to different data types in the same scope.

Example (python):

```
In [1]: a = 5

In [2]: print(type(a))
<class 'int'>

In [3]: a = 0.75

In [4]: print(type(a))
<class 'float'>

In [5]: a = "I am a string"

In [6]: print(type(a))
<class 'str'>

In [7]: a = [1,4,3]

In [8]: print(type(a))
<class 'list'>
```

Typing: strong versus weak typing

- **Strong typing:** expressions a variable can be part of depend on the type of variable. So that you cannot multiply or sum a string to an integer.

Example (python):

```
>> a = 5
>> b = 0.25
>> a+b
   5.25
>> c = "A"
>> a+c
TypeError     Traceback (most recent call last)
<ipython-input-24-e200917bc55f> in <module>
----> 1 a+c
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>> b*c
TypeError     Traceback (most recent call last)
<ipython-input-25-655218739977> in <module>
----> 1 b*c
TypeError: can't multiply sequence by non-int of type 'float'
```

- **Weak typing:** the data type can be ignored, and operations between types of different kinds are allowed. Warning: leads to errors!

Example (JavaScript):

```
4 + '7';      // '47'
4 * '7';      // 28
2 + true;     // 3
false - 3;    // -3
```

Duck typing versus nominal typing

If it walks like a duck and it quacks like a duck, then it must be a duck

- **Duck typing:** an object is of a given type if it has all methods and properties required by that type. The data type is only checked at runtime

Example (python):

```
class Duck:
    def swim(self):
        print("Duck swimming")
    def fly(self):
        print("Duck flying")
class Whale:
    def swim(self):
        print("Whale swimming")

for animal in [Duck(), Whale()]:
    animal.swim()
    animal.fly()
```

```
Duck swimming
Duck flying
Whale swimming
---------------------------------------------------------------------------
AttributeError
<ipython-input-26-731d429bd677> in <modu
     14 for animal in [Duck(), Whale()]:
     15     animal.swim()
---> 16     animal.fly()
     17

AttributeError: 'Whale' object has no attribute 'fly'
```

- **Nominal typing:** (in most compiled languages) the object is of a given type if it is declared to be; the type is known at compilation time.

Example (C++):

```
class Foo { method(input: string) { /* ... */ } }
class Bar { method(input: string) { /* ... */ } }

let foo: Foo = new Bar(); // Error when compiling!
```

The zen of python

Python has many pros as a programming language:

- Easy to use
- Easy to learn
- Easy to read (up to a certain level)
- Very good portability

However:

- Definitely **slower than C** or Fortran in many tasks
- Its flexibility can lead to errors and confusion when coding

Python is a multi-paradigm language

- Imperative
- Object oriented
- Can be used for scripting
- Is often used in data science
- Can be used for plotting and computing in the same script

Python in scientific computing

There are many common traits in the languages commonly adopted in scientific computing (**Maple, Mathematica, Matlab, S-Plus/R, Python**): simple, clean syntax, possibility to plot data directly from a script that also handles the data.

The **main advantage of python** here is that it is also possible and relatively easy to make it communicate with other languages. For instance, it can be used in combination with C, C++, Fortran. It is very portable (OS), it has a very easy and effective management of the input/output and directories (good interaction with the OS), it can be used in combination with a Graphic User Interface (GUI). It is widely used for machine learning.

Also, nowadays it is the most widely used (big community).

Python vs Matlab

They have many common features in scientific computing.
However:

- Python is more flexible and powerful
- Python is open-source and can be combined with external applications
- Matlab may be better documented
- Matlab has more routines for scientific computing
- Matlab is more stable

Very basic concepts

Declaration of variables

In python, it is extremely easy to declare a variable:

Your write the identifier (label) and assign it to a given value. *The type is guessed according to the variable declaration.*

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

```
8
```

NOTE: Python uses new lines to complete a command - i.e. each instruction has to start in a new line. Semicolons can be also be accepted (C heritage)

Identifiers

Are the names you supply for variables, types, functions, and labels in your program.
Rules for python identifiers:

- The leading character must be a letter or the character “_”
- The remaining of the identifier can be letters, numbers, “_”. Identifier names **are case sensitive!!**

Valid identifiers can be:

```
i, name_23, Pippo4, __my_name, a1b2_c3
```

Non valid identifiers:

```
2things, this is spaced, my-name, “this is spaced and in quotes”
```

NOTE: there are some reserved identifiers as:

```
and, or, else, if, for, while, def, class…
```

REMEMBER! Python variables are created when a value is assigned to them inside their scope, and are deleted when the scope is exited.

Identifiers

Are the names you supply for variables, types, fu tions
Rules for python identifiers:

- The leading character must be a letter or the chara
- The remaining of the identifier can be letters, num
  case sensitive!!

Valid identifiers can be:

i, name_23, Pippo4, __my_name, a1b2_c3

Non valid identifiers:

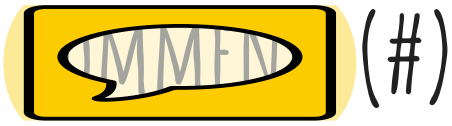
2things, this is spaced, my-name, “this is spaced ar

NOTE: there are some reserved identifiers as:

and, **or**, **else**, **if**, **for**, **while**, **def**, **class…**

```
In [1]: a = 5

In [2]: print(type(a))
<class 'int'>

In [3]: a = 0.75

In [4]: print(type(a))
<class 'float'>

In [5]: a = "I am a string"

In [6]: print(type(a))
<class 'str'>

In [7]: a = [1,4,3]

In [8]: print(type(a))
<class 'list'>
```

REMEMBER! Python variables are created when a value is assigned to them inside their scope, and are deleted when the scope is exited.



In python, you can add comments to your code with the character #

Anything following the # symbol in the same line is ignored.

Comments are important for you to remember and explain what you are doing. A well commented code is gold!

```
a=5 # number 5 assigned to the variable a
b=3 # number 3 assigned to the variable b
print(a+b) # print the result of a+b
```

```
8
```

Another kind of comment is **docstrings**, strings that are usually located as the first statements in a module, function, class, or method definition. Docstrings are meant to be used to document the code. Docstrings can also be accessed by the doc attribute on objects actually providing a handy run time help tool. *Docstring can be split on several lines.*

```
a = 5
b = 3
'''This is a docstring
which is not executed and
allows me to put comments on
multiple lines.'''
print(a+b)
```

```
8
```

Basic data types

Data types in Pyton

In python different data types are available:

- **Basic (or simple) data types:**
  - Int (long int)
  - Float
  - Complex
  - Bool
  - String
- And container data types:
  - tuple ()
  - list []
  - dict {}
  - Set
  - Frozen set
  - Object

Integers

Integer is a data type that can express any integer value that can be stored in memory. Integers in python can be written in base 10, base 8, 12, and obviously in binary.

```
Base 10: no leading 0  → n = 128     #128
Base 2:      leading 0b → n = 0b1011  #11
Base 8:      leading 0o → n = 0o15    #13
Base 16:     leading 0x → n = 0x14    #20
```

All int!!

Commas cannot be used to define an integer. Underscores are allowed instead:

```
n = 1_492       #1492
```

Floats

Floating point numbers (floats) are positive and negative real numbers with a fractional part denoted by the decimal symbol . or the scientific notation E or e.

Examples of floats: 1234.56, 3.142, -1.55, 5.2e-3, 1.2345E+2.

They can be separated with an underscore:

```
>>> f=123_42.222_013
>>> f
12342.222013
```

Before the “e” or “E” there is the mantissa, after the exponent (base 10). For instance, 1.2345E+2 is 123.45

The maximum size of a float depends on the system. The float beyond its maximum size is referred as "**inf**", "**Inf**", "**INFINITY**", or "**infinity**".
Float *2e400* will be considered as infinity for most systems.

Complex

A **complex number** is a number with real and imaginary components. For example, `5 + 6j` is a complex number where 5 is the real component and 6 is the imaginary component.

You must use `j` or `J` to identify the imaginary component. Using other character will throw syntax error.

The real and imaginary parts of a complex number can be accessed via:

```
>> x = 5+2j
>> print(x.real)
    5.0
>> print(x.imag)
    2.0
```

Bool

A boolean variable a variable that can only assume two different values:

0 (FALSE) or 1 (TRUE).

There is a specific data type in python reserved for bool, that only takes up one bit of memory.

Arithmetic operations between two numbers

+ addition

- subtraction

* multiplication

/ division

Note: an integer division leads to floats!

```
a = 5
b = 2
a/b
```

```
2.5
```

% modulus

Returns the remainder of the division of the left-hand operand by right-hand operand.

```
a = 10
b = 23
a%b
```

```
10
```

** exponent

```
a = 5.0
b = 3.0
a**b
```

```
125.0
```

// floor division

The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)

```
a = 9
b = 2
a//b
```

```
4
```

```
a = 9
b = -2
a//b
```

```
-5
```

Bitwise operators (only for int, bool)

Symbol	Name	Operation
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

```
a = 52          # 0b110100
b = 3           #    0b11
print(a&b)      #     0b0
```

```
0
```

```
a = 52          # 0b110100
b = 3           #    0b11
print(a|b)      # 0b110111
```

```
55
```

```
a = 52          # 0b110100
b = 3           # 0b10
print(a>>b)     # 0b110
```

```
6
```

Assignment operators

Symbol	Usage	Same as
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Can only be used for int and bool!!

```
a = 15
print(a)
print(a+5)
a+=5
print(a)
```

```
15
20
20
```

A BASIC IDEA OF FUNCTIONS IN PYTHON

A **function** is *a group of instructions that performs a specific task*. In many cases, it takes something in input, and returns an output.

A simple example is the **print()** function: it takes what should be printed as an argument, and it prints the associated string on the terminal or notebook.

```
>> print("Hello world!")
   Hello world!
>> a = 0.25
>> print(a)
   0.25
```

Built-in functions in Python

There are several **built-in functions** in python that can be used on the types of data described so far. Here some examples:

- `abs(x)` returns the absolute value of x
- `min(x,y,z)`, `max(x,y)` return the minimum or maximum among a set of values
- `pow(x,y)` returns x to the power y (same as x**y)
- `round(x, nd)` returns the value of x rounded to its nd digit
- `type(x)` returns the type of a variable (int, float, bool, list…)

Strings

A **string** in python is a data type consisting of a set of characters enclosed in (single or double) quotes

```
>> simpleString  = "ciao"
>> anotherString = 'This is in single quotes, and it is equivalent as the one in double quotes'
>> string_hello  = "Hello World!"
```

Strings can be *appended* with +, and a string can be *repeated* with *

```
>> first_string  = "ciao"
>> second_string = 'mondo'
>> final_string  = first_string*2+" "+second_string
>> print(final_string)
   ciaociao mondo
```

Triple quote strings (docstrings) can contain newlines and quotes:

```
>> triplequote_string  = """I am a multi-line string and
I can contain 'single', ""double"" and '''triple''' quotes
(triples only if of the other kind)"""
>> print(triplequote_string)
   I am a multi-line string and
   I can contain 'single', ""double"" and '''triple''' quotes
   (triples only if of the other kind)
```

Strings

In order to access one of the characters contained in a string, one can use the `[]` operator. The `[:]` operator gives access to a substring

```
>> simpleString  =  "ciao"
>> print(simpleString[0])
   c
>> print(simpleString[1:3])
   iao
```

Note that the indexing of the strings starts from 0!

Also note that the strings can be accessed in reverse order via the minus sign; that is:

```
>> simpleString  =  "ciao"
>> print(simpleString[-1]) #gives you the last character
   o
>> print(simpleString[-2]) #gives you the second to last character
   A
```

The `len()` function returns the length of a string.

```
>> len("ciao!")
   5
```

Strings

Strings are **immutable objects**, i.e. once initialised you cannot edit single characters by using =

```
>> simpleString  =  "ciao"
>> c[0] = "m"
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-4eefe852053a> in <module>
      1 simpleString  =  "ciao"
----> 2 simpleString[0] = "m"

TypeError: 'str' object does not support item assignment
>> simpleString  =  "miao" #this is ok, since you initialize a brand new string with the same id
```

The escaping characters start with \ and result in special characters that cannot normally be inserted in standard strings: \t (*tab*), \n (*new line*), \\ (*backslash*), \' (*quote*), \" (*double quote*), \b (*backspace*).

```
>> print("ciao\tciao")
   ciao    ciao
>> print("ciao\nciao")
   Ciao
   Ciao
>> print("\"ci\\ao\"")
   "ci\ao"
```

Strings

A literal string with a leading r or R is called “**raw string**”. Raw strings are particular strings that do not follow the usual escape rules (e.g. \n in raw strings do not start a new line).

NOTE! plot labels written in LaTex code have a lot of \ that are not escape characters! Raw strings will be very useful!

```
>> print("Hello \n world!")
   Hello
    world!
>> print(r"Hello\n world!")
   Hello\n world!
```

Strings built-in methods

Strings have a set of **built in methods** in python that can be used for manipulation, formatting and so forth

- split([sep [,maxsplit]])
- replace(old, new[, count])
- strip([chars])

```
>> s = "ciao ciao"
>> s2 = s.replace(" ","-")
>> print(s2)
   ciao-ciao
>> print(s)
   ciao ciao
```

More on strings manipulation and formatting in the afternoon notebook :)

Conversions among basic data types

It is possible to convert among different data types in python. For example

- `int()` - converts string or float to int
- `float()` - converts int or string to float
- `str()` - converts complex, int or float to string
- `complex()` - converts string, int or float to complex

Note: int are automatically converted to floats when one performs operations between the two data types. Int or float are automatically converted to complex when performing operations that involve complex numbers. In these cases there is no need for conversion.