Through Python

"Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura..."

Dante Alighieri

Syllabus

- Variables vs object
- Float representation
- The importance of syntax
- operator overloading
- built-in function vs module vs namespace
- structuring code: indentation

Variable vs Object

- A "classical" variable can bring only values, may be multiple values packed in a given structured organization :
  - a=1 ; integers
  - b="text" ; strings
  - c=[1,2,3,4,5] ; arrays
  - d={ "mad"=>1 , "max"=>2 } ; hashes
- An object can bring also **actions**:
  - a.increment_by_one() ; increment a by 1
  - b.start_with("t") ; return TRUEif b start with "t"
  - c.length() ; return the length of c
  - d.keys() ; return the list of defined keys

Variable vs Object (contd)

- Object permits to **isolate** the code for manipulating data inside a *container* with the data.
  - the final code will appear more clean with a natural separation between different functional areas.
  - debug will be more easy because objects isolate its own operations inside to itself with a clear identification of the relationship with the other part of the code.
  - It possible to improve code inside object without care of the rest of the code (if the existed relationship are holds)
- Objects are created when they are used (during the execution) and destroyed automatically by the *garbage collector* when the objects are no longer referenced without intervention by the code; the garbage collector operations takes some time and it performed at random time; these aspects can became problems in some particular applications

Float representation

- The format used by computer to represent float is derived from the format IEEE754
- It is composed by 3 parts:
  - sign: 0=plus, 1=minus
  - mantissa: a number in the range of [1..2) that can be interpreted as the key information
  - exponent: that can be interpreted as the scale of application
  - 102=+1.02*10^2 → s=0, m=1.02, e=2

Float representation(contd)

- Float format is NOT exact
  - any real number is represented by the most inner number representable by the float format.
  - Due to the fact that the minimum representable number > 0 is fixed (≅10^-20), two numbers that differs less than this quantity can not be distinguished because the difference gives 0 as the results.
- Adding float with too many different exponents can waste information
  - Due to the fact that the **precision** of the representation **is not constan**t in the whole range but decrease when exponents increase, if you sum to a very big number a very littel little number, with value less than the current precision defined by the exponent of the big number, the result of the sum is the same of the big number;

The importance of Syntax

- PAY BIG ATTENTION TO ANY CHAR WHEN YOU WRITE CODE
  - "0" (zero) is NOT "o" (lower case letter o) or "O" (upper case letter O)
  - UPPERCASE is DIFFERENT respect to lower case
  - "(" is DIFFERENT respect to "[or "{"
  - when you open some "(" or "[" or "{" usually you have to close it, in the same order.
  - Spaces cannot be used *ad libitum:* in some places, a space change meaning to a statement.
  - the order is (almost always) important: pay attention to the function arguments
  - the type of arguments of a function is IMPORTANT; if you try to append a number to a "text", you *rise* an error.
- Python 2 is NOT compatible with Python 3
  - Use Python 3! More clean, a little bit slow in some situations)

Operator overloading

- In many languages you can use the same operator to perform different actions; this is **called Operator overloading**
- Selection between different behaviours are made by check the type of the arguments to which the overloaded operator is applied:
  - In Python the "+" operator, sum, is overloaded for integers, float and string:
    - if it is applied to integers performs the sum of integers,
    - if it is applied to a float it performs the sum of float; if the other argument is an integer, then it is converted into float before the sum
    - if it is applied between strings, it performs the concatenation.

Built-in function vs module vs namespace

- In Python, there are a set of **built-in** functions, i.e. functions implemented into the *core* of the interpreter; these functions are defined in the **default namespace**: they can be called as is without any addressing prefix.
- Other function can be **imported** from external **modules**; the imported functions lie into a separate namespace: if you want to you a function, says *ceil* into module *math,* after import the module, you have to address it by an addressing prefix, *math.ceil().* If you want to use a function without the addressing prefix, you have to **import into the default namespace**

Built-in function vs module vs namespace(contd)

```
import math                 //import the module math
print(dir(math))            //print all exported functions (and  other)

a=1.2
print(math.ceil(a))         //print "2" using the namespace math

from math import ceil       //import ceil into default namespace
print(ceil(a))
```

Structuring code

Keep code clean!

- use meaningful names for variables and function
- *divide et impera*: divide big works into small steps
- *do not reinvent the wheel:* if exist a library to make a job, try to use it.
- *do not use too newer wheels*: try to use consolidated and maintained libraries
- describe your code: put (meaningful) comments at the beginning of the fuctions and the main code blocks, write the *function prototipe* at the beginning of the function.