# Lesson 1: Introduction to Python

Learning objectives: Using Python as a calculator, "Hello World" program

This is a class in Scientific Computing and its main target is to provide you with a set of concepts, tools, and tricks that will allow you to write an efficient scientific code in any language. However, teaching scientific computing requires the adoption of a language for the sake of transforming abstract lessons in practical examples that are easier to understand and remember. For this reason, the class will be taught using PYTHON as the "official" language.

This first class will be devoted to an initiation to Python.

## I: using python as a calculator

We will be using the IPython interpreter provided by Enthought. Start by clicking on the IPython icon, either on the Dock or in the Applications/Enthought folder. The program python fires up. By itself, python is not a very powerful program. We will see through the course of this semester how, thanks to its flexibility, it can be turned into a very powerful object. For the moment, let's use it as a calculator. With the "base" version of python there are only a few mathematical operations that we can perform:

1. addition: $3 + 2$
2. subtraction: $3 - 2$
3. multiplication: $3 * 2$
4. divsion: $3/2$
5. powers: $3 * *2$

A more complex calculation, involving more than one operator, can be performed, but we must pay attention to how we write it. For example, the expression:

$$1 + 2 * 3 * *4 \tag{1}$$

will give as a result 163. This is because Python gives the highest priority to powers, then come multiplications and divisions, and last addition and subtraction. Therefore the expression above is equivalent to:

$$1 + [2 * (3 * *4)] \tag{2}$$

but not to

$$(1 + 2) * 3 * *4 \tag{3}$$

or

$$1 + (2 * 3) * *4 \tag{4}$$

**Exercise I:** evaluate all the above expressions in python.

Particular attention must be paid when using the division operator /. Python uses the operator in different ways for integers and real numbers. In other words, the division between two integers will be always rounded to an integer. If asked to compute 9/5, python will return 1 as a result. A quick fix to the problem and a strong recommendation is never to use integers in your python programs, unless explicitly required. If, instead of writing 9/5 one writes 9./5., python returns the correct result 1.8.

## II: writing it all in a routine

The first step to make python become more than a calculator is to write a routine, or a series of commands that Pyton executes in sequence. For that we need an editor, i.e., a program that allows us to write expressions into a file. In this class the editor of choice will be `gedit`. However, if you are used to a different editor, you can use the one you like. Note that a word processor is not an editor. Microsoft Word, for example, cannot be used. It is important that you keep using the same editor (we will see why in a few days).

A python routine is a file containing a series of commands. It is typically saved in a file with extension `.py`. As a first step, let's write a mathematical formula in the editor $(3. + 6./7. - 2. * *3.)$. We then save it in a file called `formula.py`.

There are two ways to execute the routine. One is to enter at the prompt `python formula.py`. The other is to use the interactive version of python, called `ipython`. From now on we will always use `ipython` to execute our routines. Once in `ipython`, our routine can be executed by typing `run formula`.

If we do this, we realize that nothing happens. As a matter of fact, python computes the result of our expression, but we need to tell it explicitly to print it out if we want to. So, in order to write out the result, our program needs to contain:

`print` $3. + 6./7. - 2. * *3.$

**Exercise II:** write the above program and execute it.

## III: Strings and variables

Python can deal with much more than just numbers and mathematical operators. Let us now work on a physical example and discover many more useful concepts.

Consider the motion of a ball thrown vertically. The equation of motion reads:

$$y(t) = v_0 t - \frac{g}{2} t^2 \tag{5}$$

where we have used the origin of coordinates as the throwing point. Suppose we want to compute the position of the ball at a certain time. One thing we can do is to just write the equation as a mathematical formula with the correct numbers for each term in the equation.

However, it is better to write a flexible routine to handle the problem. A first option is something like:

```
# this routine computes the vertical position of an object

t=1.  # the time at which I want the position [s]
v0=10.  # the initial speed of the ball [m/s]
g=9.8 # the acceleration of gravity [m/s**2]

y=v0*t-g/2.*t**2

print y
```

Several new things have appeared. First, we use the character # to add comments to our routine. These are sentences in english (or any other language you can read) that explain what you are doing, the meaning of a symbol, etc. Second, we assigned a variable to the numbers. This is very useful when the same number has to be used many times in the same routine. By giving it a name and setting its value at the beginning of your routine you ensure that by changing its numerical value at the top you are propagating the change consistently throughout the whole computation. Finally, note how we have added blank lines between different logical parts of our code. There is a header with a brief explanation of what the codes does. There is a section where the variables are defined. There is a section where the result is computed. There is a section where the output is returned. Separating them with blank lines helps the readability of the routine. It's highly recommended.

We may also want to write the output with some explanation of what it is. For doing this, we introduce "strings", i.e., collection of characters. These are not variables, in the sense that no numeric value is assigned to them. Look at this new version of the program:

```
# this routine computes the vertical position of an object

t=1.  # the time at which I want the position [s]
v0=10.  # the initial speed of the ball [m/s]
g=9.8 # the acceleration of gravity [m/s**2]

y=v0*t-g/2.*t**2

print 'The position of the ball at time t=',t,'s is y=',y,'m'
```

In this new version, 'The position of the ball at time t=' is a string. Everything that lies between ' or " in python is a string. String can be concatenated with +. For example, 'ciao'+'davide' yields 'ciaodavide'. The `print` line could have been written in different ways with very similar results. For example

```
   print 'The position of the ball at time t='+str(t)+' s is y='+str(y)+'
m'
```

where we have transformed the numbers in strings and then concatenated all in a single string. Finally one can use formatted print as in:

```
   print 'The position of the ball at time t=%g s is y=%g m' % (t,
y)
```

This is called printf formatting.

## IV Line input

Finally we may want to consider that we may not want to have to edit the routine every time we want to change the initial velocity or the time at which we compute the position of the ball. On the other hand, it's pretty certain that gravity acceleration is going to remain constant for quite a while. What we can do is having a routine that, instead of having the time and velocity as numbers, asks us to type them in.

```
   # this routine computes the vertical position of an object

   t=raw_input('Time at which the computation is performed:  ')  # the
time at which I want the position [s]
   t=float(t)
   v0=raw_input('Initial velocity:  ')  # the initial speed of the ball
[m/s]
   v0=float(v0)
   g=9.8 # the acceleration of gravity [m/s**2]

   y=v0*t-g/2.*t**2

   print 'The position of the ball at time t=',t,'s is y=',y,'m'
```

The `raw_input` command asks for a user input and converts it into a string. This way you can obtain the position of the ball at different times and for different initial velocities without having to edit the routine all the times. You can convert a string that contains a number to a real (float) number with the command `float`. You can also convert a number into a string containing that number with `str`. See the above examples for the use of these two commands.

**Homework** : write a routine that asks you for your first and last name and your birth year. It calculates your age and prints out: *Hello World, my name is [yourname], I am [yourage] years old, and I wrote this Python code!!*