Python and Bioinformatics

Pierre Parutto

September 17, 2016

Contents

Chapter 1

What Is A Computer ?

Definition 1 *A computer is a machine that computes.*

The term *machine* refers to the fact that a computer is made of electronic components. The term *computes* refers to a calculation between numbers, nowadays computers are capable of very complex computations but at the hardware level all these computations boil down to a very limited set of operations on binary (0 or 1) numbers.

This chapter introduces what a computer is and briefly how it works. In order to correctly use and program a computer it is important to understand, at least very roughly, how it is able to perform all the computations. As for biological livings, nowadays computers are the result of a long and wide technological evolution.

1.1 What's inside A Computer ?

Modern computer are designed to fulfill the following criteria:

- High Performance;
- Low Cost;
- Strong Reliability;
- Low Power Consumption (since recently).

1.1.1 Electronics components

A picture of the inside of a (my) standard personal computer is presented in Figure 1.1. The main electronics components present are the following :

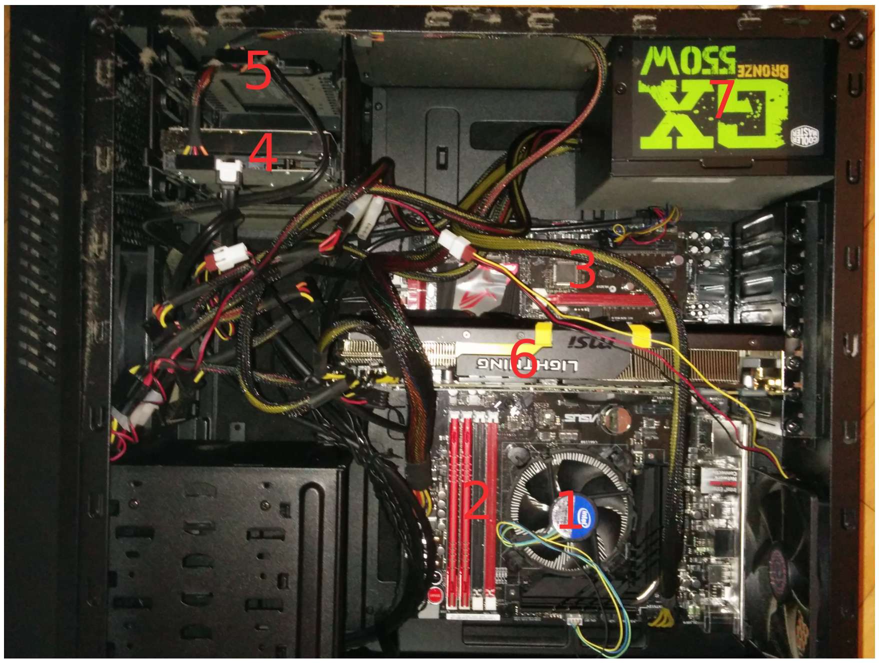


Figure 1.1: The insides of a classical personal computer.

1. The Central Processing Unit (CPU) is the core of the computer, in charge of all the computations (except for the display if a graphic card is present);

2. The Random Access Memory (RAM) is a temporary memory used by the programs to store the data they need. This memory is volatile, all the content is lost if the power is turned off. It is faster to access than the ROM memory but has a capacity of ”only” a few gigabytes;

3. The Motherboard is mostly cables and plugs that orchestrates the communications between all the components.

4. A mechanical hard drive connected through a SATA wire. It is a non-volatile memory (its content remains when you turn off the computer). It can store a large amount of data, of the order of the terabyte now, it is slower to access than the RAM.

5. Another hard drive of the Solid State Drive (SSD) type, a flash memory, faster than the mechanical one but limited in the number of writing operations before being unusable. It can stores in the order of hundreds of gigabytes.

6. The Graphic Card (GC), a piece of hardware specifically created to make computations related to the display. It is composed of many processing

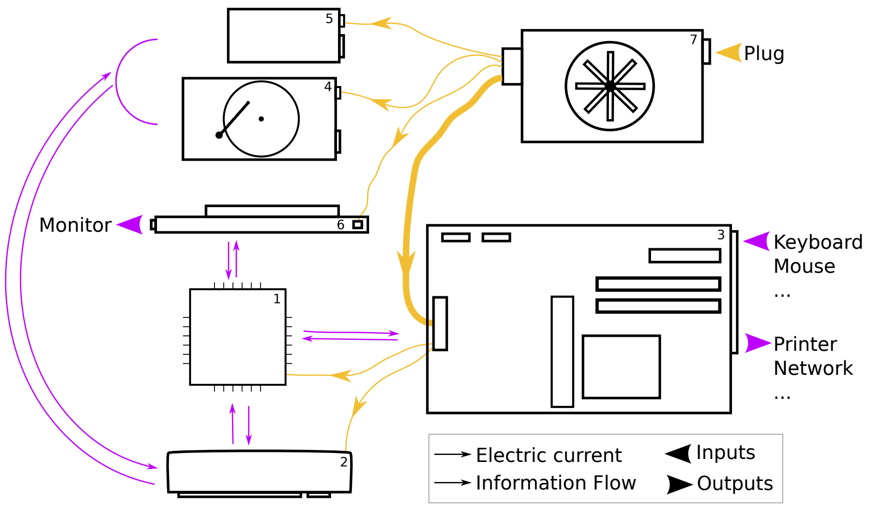


Figure 1.2: Electric current and information flow in-between the principal components of a computer and with the exterior. Component labels are as in 1.1.

units that allow to make many computations in parallel.Mostly used for displaying videos and rendering video games.

7. The power supply allows to provide electrical current to all the electronic components of the computer.

Among other things on this picture are some fans and the computer case that are needed to cool the components. This is a task of prime concern as they generate a lot of heat during their functioning mainly due to joule's effect. Finally you can also see a lot of cables.

1.1.2 Information Flow

In a computer the information flow respects the cyclic route presented in Figure 1.2:

1. A piece of information arrives to the motherboard through an external source : the mouse or keyboard, the network, . . .;
2. It is processed by the CPU that may need to fetch items in memory either from the RAM or any disk;
3. Once processed, the processed piece of information can be stored to a disk, displayed to the monitor through the graphics card, sent to another device through network via the motherboard, . . .
4. Go back to step 1;

This cycle goes on forever while your computer is on.

Remark

Most of the time, the CPU does not read directly to the hard drives. When a piece of information from a HD is needed, it is first loaded into the RAM, and then accessed from there by the CPU.

1.2 The Operating System

The Operating System (OS) is a **HUGE** piece of software that starts a little bit after the computer is turned on (before that things are handled by the BIOS) and stops when you turn off the computer. It is the bridge between the hardware (electronics components) and the softwares. Its main tasks along with the different kind of interactions between the different actors are presented in Figure 1.3. It is composed of the following modules:

- Drivers: Allows programs to access to the hardware (graphics card, hard drive, ...).
- Program Manager: Run all the needed programs, ensure a fair share of CPU usage and other resources (called scheduling).
- Window Manager: Provide a nice interface to the user and allow program to create windows of their own.
- File system: Organize the files and directories from the users.

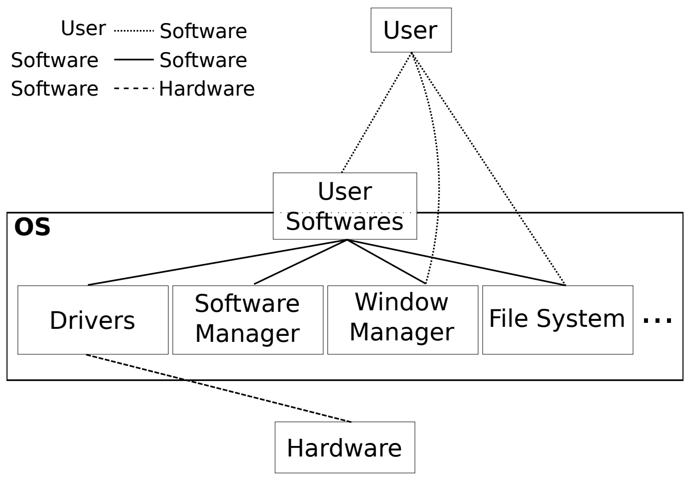


Figure 1.3: Typical components of an Operating System (OS) and their interactions with users, softwares and hardwares. Internal interactions in the OS are not represented.

1.2.1 Different Operating SystemS

There exists many different families of operating systems, the three most famous being Microsoft Windows, Apple MacOs and GNU Linux.

Windows is the oustider here, possessing its totaly own code whereas MacOs and Linux share some common code base. Therefore MacOs and Linux OS are in some way pretty similar and its easier to go from one to the other than with Windows. The main difference in MacOs being mostly located in the window manager.

That being said, differences between OS can concern any of their modules : the filesystem, the drivers, ... Programs interacts mostly with the operating system and thus are very exposed to different behaviors between OS. It is thus safer to consider by default that some software program for one OS will not work on another.

Remark

A simple example of different behaviors between OS lies in the different ways of for example writing in a file on the HD between windows and linux.

Chapter 2

Programming Languages

2.1 Programming Languages

Definition 2 *A programming language is a language designed to give orders (to program) a machine.*

This section quickly presents computer languages in the light of natural languages, then the two big families of programming languages: compiled and interpreted languages and finally gives a more specific overview of the Python language.

2.1.1 Natural Languages and Programming Languages

Definition 3 *A language is a set of words (representing ideas) and a set of rules (called grammar).*

Natural and programming languages are very similar except that the last one is made to give order to machines.

2.1.1.1 Evolution Of Natural Languages

Natural languages evolve through time, as an example compare this *sonnet 2* by William Shakespeare (1564-1616):

> When forty winters shall beseige thy brow, And dig deep trenches in thy beauty's field, Thy youth's proud livery, so gazed on now, Will be a tatter'd weed, of small worth held: Then being ask'd where all thy beauty lies, 5 Where all the treasure of thy lusty days, To say, within thine own deep-sunken eyes, Were an all-eating shame and thriftless praise. How much more praise deserved thy beauty's use, If thou couldst answer 'This fair child of mine 10 Shall sum myg count and make my old excuse,' Proving his beauty by succession

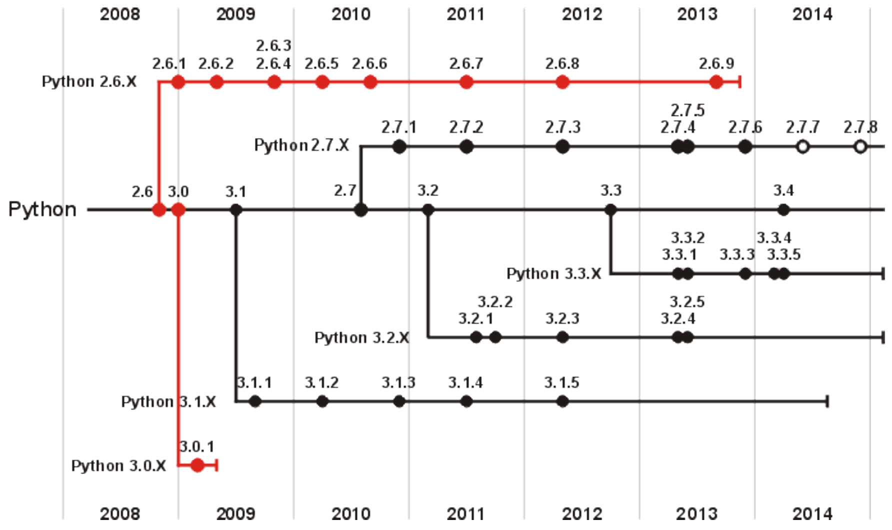


Figure 2.1: Timeline of Python versions, red dots correspond to obsolete versions (from `http://www.mclibre.org/`)

> thine! This were to be new made when thou art old, And see thy blood warm when thou feel'st it cold.

To this extract from *The Hichhiker's Guide to the Galaxy* by Douglas Adams (1952-2001):

> The house stood on a slight rise just on the edge of the village. It stood on its own and looked over a broad spread of West Country farmland. Not a remarkable house by any means - it was about thirty years old, squattish, squarish, made of brick, and had four windows set in the front of a size and proportion which more or less exactly failed to please the eye.

As we can clearly see from these text, the english language from 500 years ago is alike but not the same as the one written nowadays.

2.1.1.2 Evolution Of The Python Programming Language

Python was created by Guido Van Rossum in 1989 at the Centrum voor Wiskunde en Informatica in Amsterdam. It is an **open source** language: all the characteristics of the language are available freely online. It is also a community driven language, as Python evolves following the propositions of the community.

As for natural languages, the Python language evolved a lot throughout these last 25 years. Figure 2.1 presents the different versions of Python released since its creation.

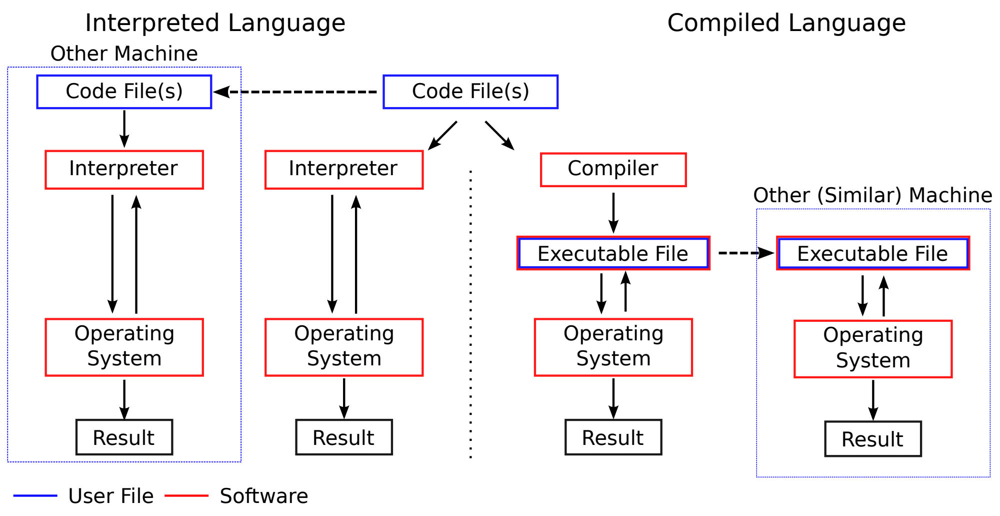


Figure 2.2: Difference between interpreted and compiled languages.

Warning

In this class we will use the **versions 3.4** of Python.

2.2 Compiled vs Interpreted Languages

In the world of programming languages, there exist two big families: compiled vs interpreted languages. The difference between the two lies in how they produce code understandable by the machine:

- A compiled language uses a compiler software to translate code files into an executable file that can be directly run by the operating system. The process of running code files through a compiler is called compilation. The executable file obtained can then be run independently of the compiler on the computer or any computer with similar OS and CPU.
- An interpreted language uses an interpreter software that both translate and run code files at once. The process of interpreting a code file is called interpretation. No executable file is produce, the code is simply translated and ran right after. If you want to run such code file on a machine it needs to have the interpreter software installed on it.

The Figure 2.2 summarizes the differences between the two families of languages.

The advantages and downsides of of each family are:

- Compiled languages:

+ Fast, the created software is built for the machine;
+ Can distribute directly the compiled file to other similar (same type of CPU and OS) computers;
- Cannot be distributed to machine with different OS or CPU.

- Interpreted languages:

  + Very easy to share to any type of computer (it just needs to have an interpreter installed.
  - Slower because the interpreter translates the instruction and then run them.

Python is an interpreted language which means that you need to have the Python interpreter on your machine in order to be able to run Python code.

2.3 Python Language vs Python Software

The word Python is used to refer to two specific notions:

- **The Python language**: a programming language (as for spoken/written languages) is only a set of rules. The Python language possesses a grammar that can be found here, these rules must be followed in order to do a valid Python ”sentence”. In addition to the grammar, Python also comes with a set of known words called builtins that can be found here.
- **The Python software** Python is also the name of the software that is capable of translating Python (the language) codes into orders understandable by your CPU. There exist different software capable of interpreting Python codes such as Jython or pypy.

Warning

In the remaining of the class, when I say Python, I wont usually specify whether it is the language or the software I am speaking of. It should be clear from the context.

2.4 Using the Python Software

The Python Software can be used in two different ways: line by line (like a dialog) or using code files (like a monologue). To write codes in Python we will use the Spyder Integrated Development Environment (IDE), that allows us to write and execute code in the same software.

Remark

There exist many different IDEs to write Python code. You can also just use any text editor and then use the Python interpreter directly on files.

2.4.1 Line By Line Mode

The line by line mode provides a direct access to the Python interpreter in which you can type a command and directly gets the result. In this mode, Python displays three chevrons >>> at the beginning of the line to indicate that it awaits for a command. Once a command is provided Python (the interpreter) evaluate it and display the result (if any) on the following line. For example:

```
>>> 5
5
>>> 5 + 1
6
```

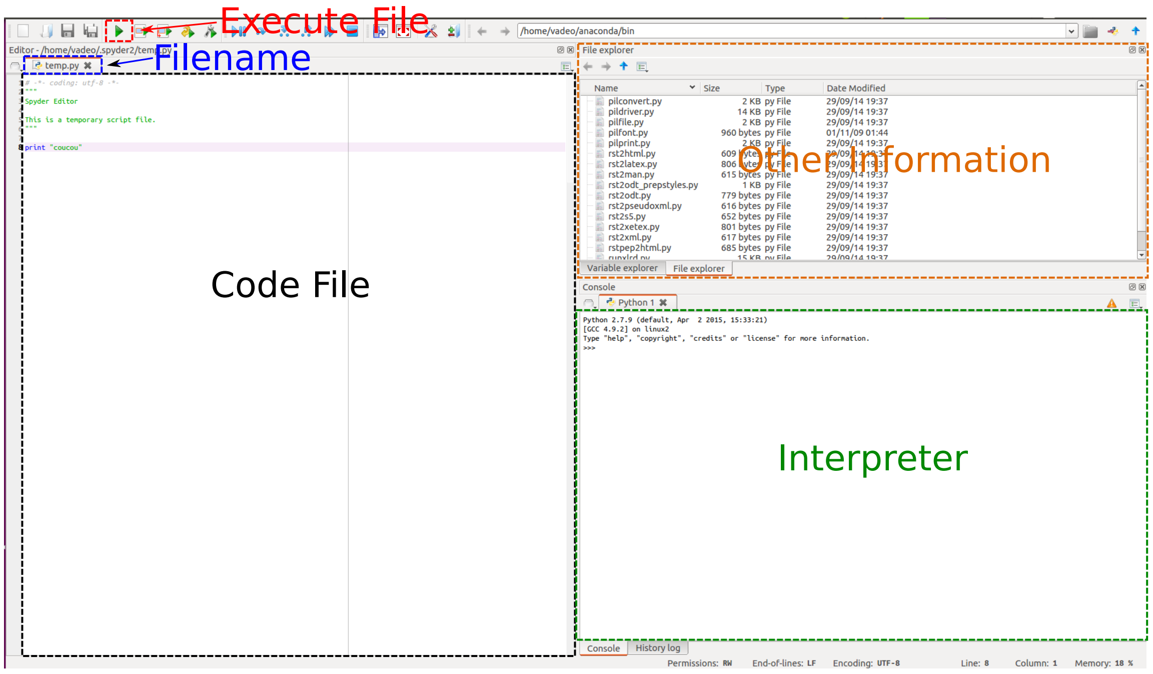
I will use this mode extensively in the class as it allows to quickly present the behavior of some Python command. On the bottom right panel of Figure 2.3 you can see the Python interpreter inside the Spyder software.

2.4.2 File Mode

In this mode, you provide Python directly with one (or multiple) Python code file(s) and the interpreter evaluate them. A code file is simply an ensemble of python commands with one command by line. On the left panel of Figure 2.3 you can see the python code file named `temp.py` opened inside the Spyder software. To interpret this file in Python one needs to click on the green triangle, in red on the top of the picture.

Remark

To interpret a file Spyder uses Python interpreter by using the command `runfile` that makes Python interpret a code file.



```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

print "coucou"
```

Name	Size	Type	Date Modified
pilconvert.py	2 KB	py File	29/09/14 19:37
pildriver.py	14 KB	py File	29/09/14 19:37
pilfile.py	2 KB	py File	29/09/14 19:37
pilfont.py	960 bytes	py File	01/11/09 01:44
pilprint.py	2 KB	py File	29/09/14 19:37
rst2html.py	609 bytes	py File	29/09/14 19:37
rst2latex.py	806 bytes	py File	29/09/14 19:37
rst2man.py	615 bytes	py File	29/09/14 19:37
rst2odt_prepstyles.py	1 KB	py File	29/09/14 19:37
rst2odt.py	779 bytes	py File	29/09/14 19:37
rst2pseudoxml.py	616 bytes	py File	29/09/14 19:37
rst2s5.py	652 bytes	py File	29/09/14 19:37
rst2xetex.py	801 bytes	py File	29/09/14 19:37
rst2xml.py	617 bytes	py File	29/09/14 19:37
rstpep2html.py	685 bytes	py File	29/09/14 19:37
runxlrd.py	15 KB	py File	29/09/14 19:37

```
Python 2.7.9 (default, Apr  2 2015, 15:33:21)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 8 Column: 1 Memory: 18 %

Figure 2.3: The Spyder development Environment with its principal functions.

Chapter 3

First Steps In Python

From now on we will start programming, basically it consists in manipulating values. This simple goal hides many concepts that we will define throughout the class. We will start first by defining what is a value and its associated type; We will then see the expressions obtained by using the logical, arithmetic or comparison operations on values; Finally we will define what a variable is.

3.1 Values and Types

3.1.1 Values

Definition 4 *A value is a piece of information provided to the program.*

Values can be acquired in different ways:

- Directly provided by the programmer in the code;
- Provided by the user when using the program;
- Read from another source of information (file, network, . . .).

3.1.2 Types

In the first class we have seen that from the point of view of the CPU everything is only 0 and 1, hence the interpretation of a value entirely depend on the meaning it was given.

Definition 5 *The type of a value provides the information of how this value should be interpreted.*

The basic types used in programming are the following:

- Boolean (`bool`): represent the values `True` and `False`;

- Integer (`int`): represent a positive or negative discrete number ($\in \mathbb{Z}$);
- Floating point number (`float`): represent a decimal number ($\in \mathbb{D}$);
- Character (`char`): represent an alphanumeric character (a-z, A-Z, 0-9, punctuation).

Warning

- In Python the values of Boolean type have a capital letter as their first characters: **T**rue and **F**alse.
- In programming the capital characters are distinguished from the small ones: `"A"` is different than `"a"`.

Remark

- Python follows the American notation for floating point: you must use a dot . to separate the integer part from the decimal one: 5.4534.
- Actually in Python characters alone does not exist, instead python provides the type `string` (of characters). To simulate a character, we will at the beginning only work with strings of length 1.

To know the type of a variable, Python provides the function `type`.

3.1.3 Examples - Values And Types

In the following I enter some values inside the interpreter and I look at some types.

```
>>> 5
5
>>> "a"
a
>>> 5.5
5.5
>>> type(3)
<class 'int'>
>>> type(-3)
<class 'int'>
>>> type(-3.0)
<class 'float'>
>>> type("-3.0")
<class 'str'>
>>> type(True)
<class 'bool'>
```

Warning

In python3, the type is called ”class”, do not let the term confuse you, a class is a special kind type. We will see more about classes during the second semester.

3.2 Expression and Operations

One can obtain a new value by making operations on values. Think of the addition:

```
>>> 5 + 2
7
```

In this code I provided two values: 5 and 2 and I created a third one (I did not enter it) 7.

3.2.1 Expression

Definition 6 *An expression is a piece of Python code that creates a value.*

An expression is thus either:

- A value;
- An operation between one or more expression(s);

We will see the basic type of operations that can be done in Python.

3.2.2 Logical Operations

These operations work on Boolean values, and are similar to the one we have seen in the first class. Let `a` and `b` be to Booleans, the most common logical operations are:

Name	Symbol	Truth Table		
		`a`	`not a`	
Negation	`not a`	True	False	
		False	True	
		`a`	`b`	`a or b`
		True	True	True
Conjunction	`a and b`	True	False	True
		False	True	True
		False	False	False
		`a`	`b`	`a and b`
		True	True	True
Disjunction	`a or b`	True	False	False
		False	True	False
		False	False	False

3.2.3 Arithmetic Operations

These operations work on integer and floating points values, they are the natural operations on numbers. Let `a` and `b` be two integers or floats, the common arithmetic operations are:

Name	Symbol
Addition	`a + b`
Subtraction	`a - b`
Division	`a / b`
Multiplication	`a * b`
Exponentiation(i.e. a^b)	`a ** b`
Integer division	`a // b`
Modulus	`a % b`

The first five operations are known, let us focus on the last two that you might not know:

- The integer division also called Euclidean division, is a division between integer which result is also an integer.
- The modulus `a % b` provides the remainder in the Euclidean division of `a` by `b`.

Remark

The integer division and modulus are linked by the following relation:

$$a = b \times q + r$$

Where $a, b, q, r \in \mathbb{N}$, a is called the dividend, b the divisor, q is the quotient and r the remainder.

In this expression `q = a//b` and `r = a % b`.

3.2.4 Comparison Operations

Finally, the following operations allow to compare two values, the comparison operations create a value of Boolean type. Let `a` and `b` be two values:

Name	Symbol	True if
Equality	`a == b`	`a` and `b` have the same value
Difference	`a != b`	`a` and `b` have different values
Less than	`a < b`	`a` is less than `b`
Greater than	`a > b`	`a` is greater than `b`
Less than or equal	`a <= b`	`a` is lesser than or equal to `b`
Greater than or equal	`a >= b`	`a` is greater than or equal to `b`

3.3 Evaluation Of Expressions

Definition 7 *We call evaluation the action of computing the value represented by an expression.*

Expressions are evaluated from left to right (as we usually do), following specific priorities between operator. The priorities you know in calculus are extended to include all the operators we have seen previously and are as follows from the lowest (evaluated last) to the highest (evaluated first):

Priority	Operators
1	`or`
2	`and`
3	`not`,
4	`==`, `!=`, `>`, `<`, `>=`, `<=`
5	`+`, `-`
6	`*`, `/`, `%`, `//`
7	`**`
8	`()`

Warning

For two operators with the same level of priority, the first come first evaluated rule applied. For example:

```
>>> 4 * 5 % 5
0
```

```
>>> 4 % 2 * 5
20
```

In the first expression, `4 * 5` is evaluated first, whereas in the second expression `4 % 2` is evaluated

Remark

The parentheses are not operators but are used to enforce the evaluation of the enclosing expression before the others.

Chapter 4

Variables

4.1 Variables

Definition 8 *A variable is a name associated to a value.*

Assignment Operator To define a new variable, Python provides the assignment operator `=` that works as: `variableName = value`. With `VariableName` any **valid variable name** you want. The `variableName` is also called the Left Hand Side (LHS) and the `value` the right hand side (RHS) of the assignment operation.

Example The following cases are all valid variable assignments:

```
>>> a = 5
>>> b = 5 + 7.0
>>> c = b
>>> d = a + 2
>>> e = a + d
>>> f = True
```

Remark

In the example, Python provided no answers to the commands, this is because an assignment does not evaluate to any value.

4.1.1 Using Variables

To use a variable in an expression, just put its name and it will be replace by its associated value.

Example The following cases use variables in computations:

```
>>> a = 5
>>> b = a + 5
>>> a + b * 2
25
>>> a = a + 1
>>> a
26
```

4.1.2 Type associated to variables

Definition 9 *The type of a variable is the type of the value associated to it.*

If the value associated to the variable changes to another type then the type of the variable also changes.

4.1.3 Python Memory

The Python interpreter, like a human being, possesses a memory. At the beginning of a program, this memory is filled with some common names corresponding to keywords and built-ins:

- Keywords are a small specific ensemble of names reserved by Python, such as `for`, `while`, ... These are names needed by the Python interpreter to be able to correctly interpret Python codes. **Keyword names cannot be redefined.**
- Built-ins are an ensemble of variables and functions that are defined at the beginning of all Python programs. Most of these built-ins contain very commonly used values and functions. **built-in names can be redefined**. A list of all the built-in functions can be found here while the built-in variable can be found here.

When encountering an assignment Python distinguishes two cases:

- **The variable name is not known to Python**: then Python evaluates the RHS, add the LHS to its memory and associate to it the value of the RHS.
- **The variable name is already known to Python**: then Python evaluates the RHS and replace the current value by this new one.

Warning

The RHS is evaluated before associating the value to the LHS name. Hence one can write code like:

```
>>> a = 5
>>> a = a + 1
>>> a
6
```

Here, on line 2, Python evaluates the RHS using the current value of `a` which is 5 and then assigns the result of the expression to `a`. Note however that `a` needs to be declared, hence the following **does not work**:

```
>>> a = a + 1
NameError: name 'a' is not defined
```

Warning

When using variables, there is one simple and natural rule to respect: **do not use variable names that you did not define earlier in the program**.

Remark

There exists the keyword `del` which works as: `del nameVariable`. It deletes (makes Python forget) the variable `nameVariable` from Python's memory.

4.1.4 Variable Naming

The following names are the keyword reserved by Python, **you cannot have** variables with these names:

```
'and', 'as', 'assert', 'break', 'class', 'continue', 'def',
'del', 'elif', 'else', 'except', 'exec', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not',
'or', 'pass', 'print', 'raise', 'return', 'try', 'while',
'with', 'yield'
```

Except for these names, variable names:

- **Can only contain** alphanumeric characters (letter + digit) and the character _ (ex: `a`, `zzZzZz`, `A223`, `a_b`);
- **CANNOT** start with a number (ex: `3a`, `3A3`, `3`).

Warning

Except for the previous conditions, you are free to chose **any** variable name you want. It is however a good practice to name variables depending on what they represent. I strongly advise you to do that if you want to keep your programs readable.