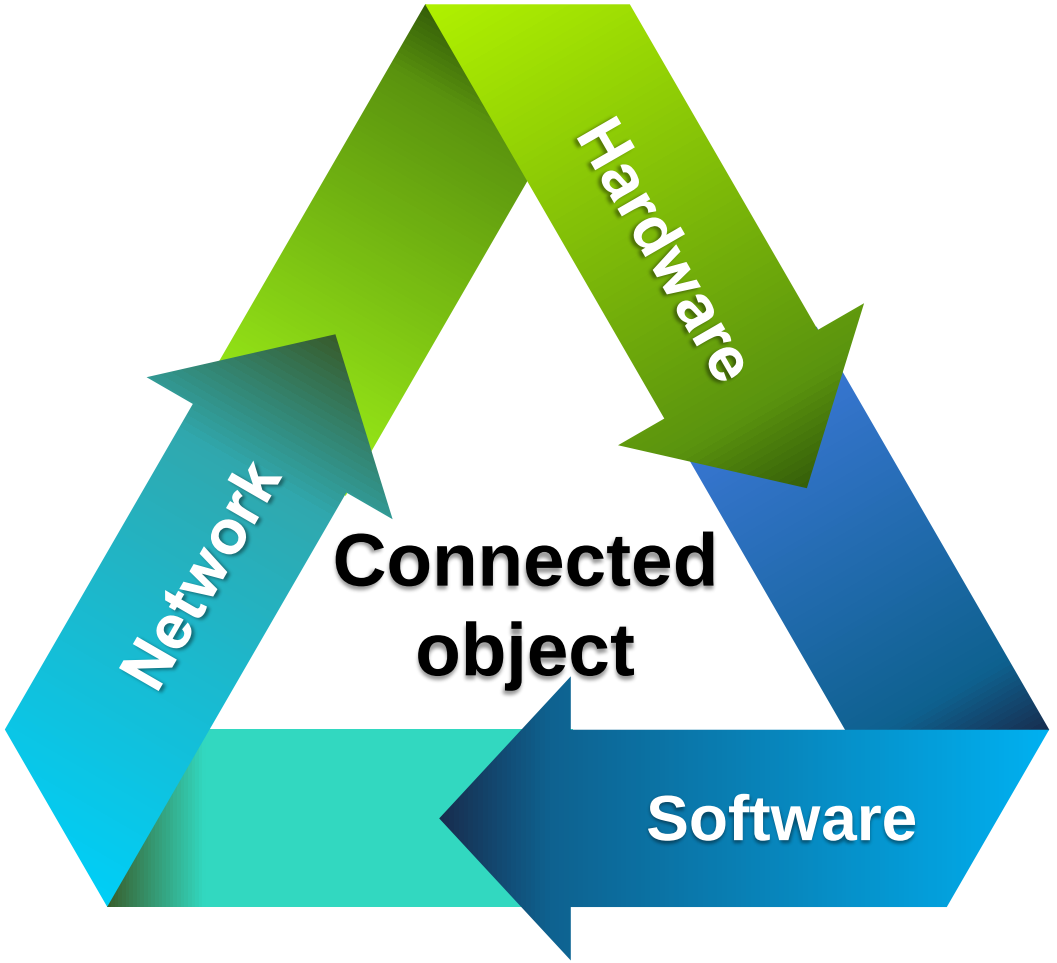
Software and Python Programming

Jad Nassar, PhD


Hardware
Network
Connected object
Software

Software definition

- A software program is a set of sequences of instructions that are understandable by a machine.

- The implementation of the software instructions is called its execution.

- **Embedded software:**

A micro controller is slow and limited. There is only room for one software, yours.
So, your software must do its job and manage all the hardware aspects at the same time: RAM management, storage memories, processing speed, battery, etc...

If you want to do x things at the same time you use ?

Software definition

- On the other hand, a computer like the Raspberry has enough computing power and memory to run dozens of programs at the same time.
- Software can technically be classified as **system software** or **application software** depending on how it interacts with the hardware.
- This is not just an abstract view: the computer works differently when it runs one or the other of these categories of software:
    - On a microcontroller a software is necessarily ... system.
    - On the other hand, on a computer we have...

❑ Operating System

Operating system: Origin, definition

- There are hundreds of different operating systems, more or less recent, popular or complex.
- On the "Desktop" market Windows dominates with >95% market share for decades.
- On "Mobile" it is Google's Android (60%?) that leads the way, followed by Apple's iOS (30%).
- On the Server market, it's Linux and Windows Server.
- For an average user, it is very difficult to define precisely where an operating system starts and stops [in other words, what is an application software or system] because each manufacturer (Microsoft, Apple, Google...) has its vision.

Operating system: Origin, definition

- From a more scientific point of view, an operating system (or OS) is a set of software called "system software" that has two main roles:
  - To provide an extended machine: The OS acts as an abstract or virtual machine that takes application programs as input and processes it after interacting with the hardware. Since this behaves like **a machine taking some input and producing some output** hence it is also known as an extended machine.
  - To manage resources: We can say that the OS must manage all the elements of a very complex system. Modern computers are made of one or more processors, multiple memories, mouse, one or more network cards, screen...
    The OS must manage in an equitable (or not) and optimal way the allocation of resources to the different applications or systems that request them.

Why Python?

- Python is portable, not only on the different Linux variants, but also on proprietary Operating Systems: Mac OS, Windows....
- Python is free, but it can be used without restriction in commercial projects.
- Python is suitable for scripts with about ten lines as well as complex projects with tens of thousands of lines.
- Python syntax is very simple and, combined with advanced data structures (lists, tables...), leads to programs that are both very compact and very readable.
- Python is extensible: it can easily be interfaced with existing libraries.
- According to its supporters, for equal functionality, a Python program is often 3 to 5 times shorter than an equivalent C or C++ (or even Java) program, which generally represents a development time which is 5 to 10 times shorter.

Our usage

- We will use the execution of Python files using script files with the extension.py that we will launch from Bash.

- There are several versions of python, incompatible with each other. We will limit ourselves in this course/Lab to **Python >3.X**

First step

- **nano text editor**

Introduction

- In this Course/Lab you will:
  - Connect to the same server as Lab1: 10.34.161.21
  - Use the same username/password as in Lab1.
- The server runs on Linux, and you will use the Bash shell to launch a ***nano*** text editor. This editor will be used to create your python scripts.

Nano text editor

- A text editor is a program that allows you to modify plain text files, without formatting (bold, italic, underlined...).
- Under Windows, you have a very basic text editor: Notepad.
- Under Linux, you can choose between Nano, Vim, Emacs and many others, knowing that at least one of these is installed by default on most distributions.
- Text editors are used... to edit text! For example, configuration files, program source code, etc.

Using the Nano text editor

How to install Nano?

- The ***Nano*** editor is available by default in all of the most popular Linux distributions and you can run it with a simple command:

```
nano
```

- The above command will simply open a new file. You can type in the window, save the file and exit.

Using the Nano text editor

How to open a new file and give it a name using Nano?

- Although the simple fact of running nano is acceptable, you can give your document a name before you start. To do this, simply add the file name after the nano command.
  The main command we will use is then:

```
nano filename
```

- After executing the command, you will be directed to the editor window, where you can now use and edit your text with the nano text editor. Use the arrow keys on your keyboard to move the cursor.
- At the bottom of this window, you will see some of the shortcuts that can be used with the nano editor. The symbol ^ means that you must press CTRL +[Key] to launch the selected command.
- You can find in the next slide some of the main commands. You can consult the complete list on the internet.

Using the Nano text editor

Command	Explanation
CTRL + A	Go to the beginning of the line
CTRL + E	Go to the end of the line
CTRL + O	To save your file. When this command is used, you will be prompted to change or check the name of the desired file and after pressing Enter, it will save your file.
CTRL + X	Exits the Nano text editor. If you have made changes to the file, it prompts you to save.

Using the Nano text editor

How to open an existing file using Nano?

You can use the same command as in the previous slides to open an existing file.
Reminder :

```
nano filename
```

- All you have to do is to run nano with the path to the file you want to open.
- To be able to edit the file, you must have the permissions, otherwise it will open as a read-only file (assuming you have the read permissions).

Using the nano text editor with Python

- In order to facilitate code reading, syntax highlighting is practically essential. The keywords of the languages are highlighted, the comments a little less, etc.... Nano recognizes the python code as such and colors the python keywords. As soon as Nano opens a file ending in ***.py*** it will use this syntax highlighting.
- You can configure Nano text editor system using the ***nanorc*** configuration file.

- Open this nano configuration file by typing :

```
nano ~/.nanorc
```

- Then add the following text to the file:

```
set nowrap
set autoindent
set tabstospaces
set tabsize 4
```

- Save and exit nano

Signification:

set nowrap: do not wrap long lines.

set autoindent: indent new lines to the previous line's indentation. this is also useful when editing source code.

set tabstospaces: convert typed tabs to spaces.

set tabsize 4: set width of a tab

Python Programming

Why Python?

- Python is portable, not only on the different Linux variants, but also on proprietary Operating Systems: Mac OS, Windows....
- Python is free, but it can be used without restriction in commercial projects.
- Python is suitable for scripts with about ten lines as well as complex projects with tens of thousands of lines.
- Python syntax is very simple and, combined with advanced data structures (lists, tables...), leads to programs that are both very compact and very readable.
- Python is extensible: it can easily be interfaced with existing libraries.
- According to its supporters, for equal functionality, a Python program is often 3 to 5 times shorter than an equivalent C or C++ (or even Java) program, which generally represents a development time which is 5 to 10 times shorter.

Our usage

- We will use the execution of Python files using script files with the extension.py that we will launch from Bash.

- There are several versions of python, incompatible with each other. We will limit ourselves in this course/Lab to **Python >3.X**

- Variables
- Assignment
- Printing to the screen
- Operators

Variable

- A variable appears in a programming language under a name, but for the computer it is a reference to a specific place in the memory.
- At this location a specific value is stored. This is the data itself, which is therefore stored as a sequence of binary numbers.
- To distinguish these various possible contents from each other, the programming language uses different types of variables.

Variable name

- A good programmer must ensure that his program is readable!
- He must choose explicit and concise variable names which is essential while respecting the rules: a variable name is a sequence of letters (a→z, A→Z) and numbers (0 → 9), **which always begins with a letter.**
- Only one other character is possible _ (underscore)
- It is case sensitive (toto is different from Toto).

Variable types

- As already mentioned, variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

- Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning values to variables

- Python variables do not need explicit declaration to reserve memory space.
- The declaration happens **automatically** when you assign a value to a variable.
- The equal sign (=) is used to assign values to variables.
- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

Example:

```
counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string
```

This is a commentary,
we will get back soon to this

For the curious ones:

In Python, several operations take place at the same time when the instruction counter = 100 is written per example.

- Create and store the name of variable counter.
- Assign an integer type to it
- Create and store the value in the dynamic typing memory: Python infers (=deduces) the type of the variable.
- Establish a link between the reference variable and the referenced value.

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously.

Example:

```
a = b = c = 1
```

Here, the value 1 is assigned to the variables a, b and c.

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Printing to the screen

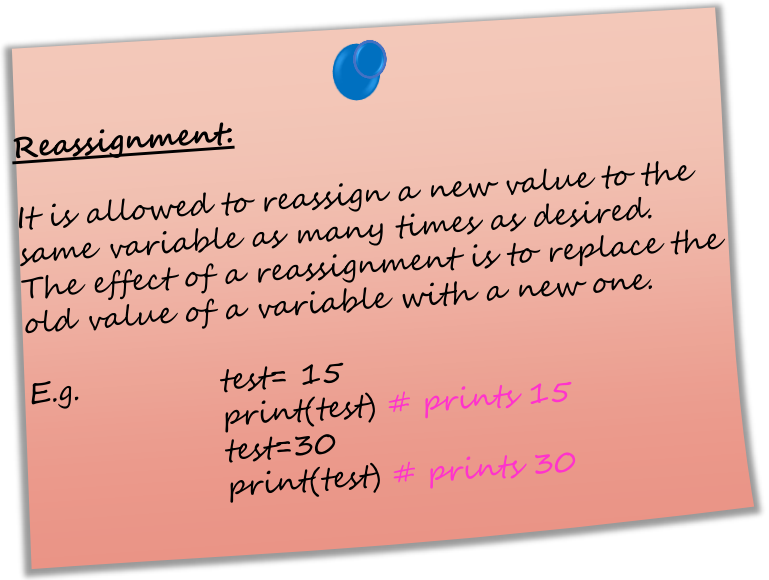
- To know the value of a variable in Python, we use the ***print()*** function.
- This function converts the expressions you pass into a string and writes the result to standard output (the screen).

Example:

```
print "Python is the best"
```

This produces the following result on the screen:

Python is the best.



Operators

- Operators are the constructs which can manipulate the value of operands.
- Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Arithmetic operators:

- Python has the 4 basic arithmetic operators +,-,*,/
  In addition, there are operators of Floor division // exponent ** and modulus %

Example 1 :

```
a, b = 7.3, 12
y = 3*a + b/5
```

Comparison operators:

- These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Operator	Meaning
==	Equality test
!=	difference
>	Greater
<	Less
>=	Greater then or equal
<=	Less then or equal

Example 2:

```
h, m, s = 15, 27, 34
print("number of seconds= ", h*3600 + m*60 +s)
```

The easiest way to print multiple values on the same line is to follow print with those values separated by commas.

Operators

Boolean operators:

a	not a
False (0)	True (1)
True (1)	False (0)

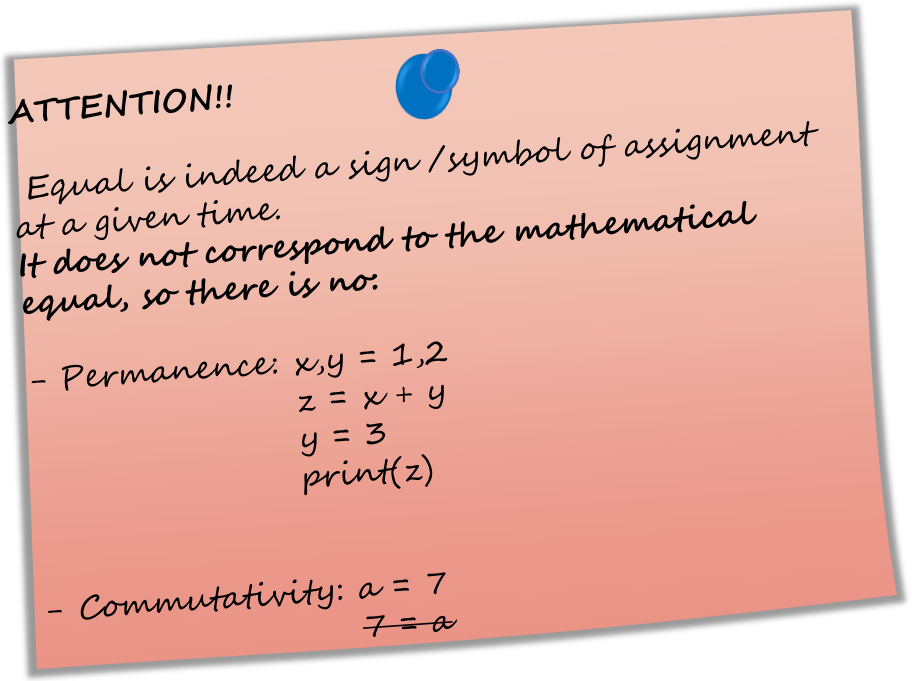
a	b	a or b
False (0)	False (0)	False (0)
False (0)	True (1)	True (1)
True (1)	False (0)	True (1)
True (1)	True (1)	True (1)

a	b	a and b
False (0)	False (0)	False (0)
False (0)	True (1)	False (0)
True (1)	False (0)	False (0)
True (1)	True (1)	True (1)

Operators priority

- In python, the priority rules are the same as those taught to you in the mathematics course.
- You can memorize them using an easy “trick”, the acronym: **PEMDAS**

- **P** Parentheses first
- **E** Exponents (i.e. Powers and Square Roots, etc.)
- **MD** Multiplication and Division (left-to-right)
- **AS** Addition and Subtraction (left-to-right)



Spaces and Commentaries

- Spaces and comments are normally ignored. Apart from those used for indentation, at the beginning of the line, spaces placed inside instructions and expressions are almost always ignored (unless they are part of a string).

- The same applies to comments: they always start with the hashtag character (#) and extend to the end of the current line.

Example:

```
# This is a commentary
# These two lines are invisible
```

Exercises

Let's test all of this!

For all of the following exercises, create a TP_Python folder in your home folder on bash, then create one file per exercise with nano and write your code. (`nano exercice_name.py`)

To test your code, you must type under bash :

```
python3 exercise_name.py
```

Exercises

- 2.1 Describe as clearly and completely as possible what is happening in each of the three lines of the example below:

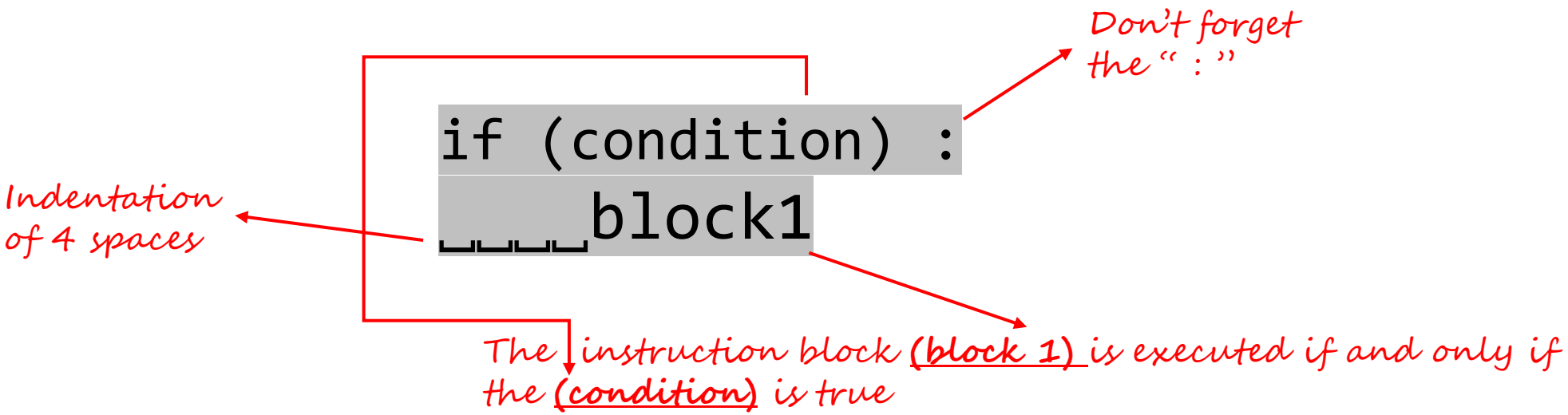
```
width = 20
height = 5 * 9.3
width * height
```

- 2.2 Assign the respective values 3, 5,7 to three variables a, b, c.
  Perform the operation a-b//c and display the result.
  Interpret the obtained result.

- Conditional Statements
- Instruction Blocks
- Loops

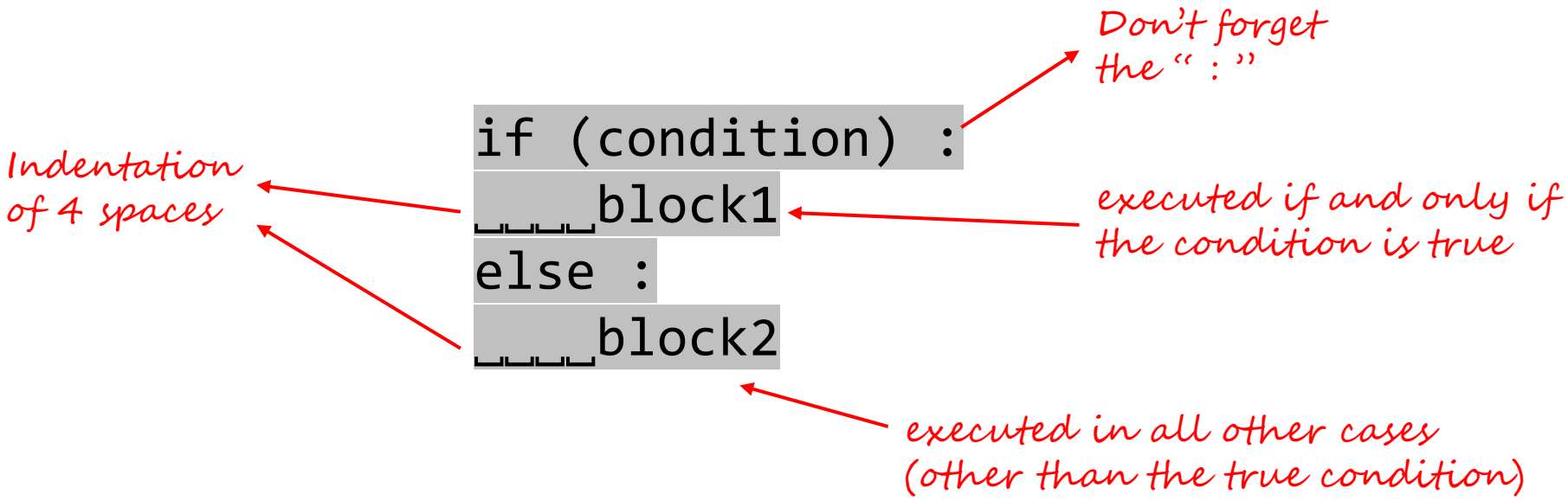
Conditional statements

- The easiest conditional statement is ***if :***



Conditional statements

- The conditional instruction if can be completed by the else. That's the alternative. It is used as follows:



The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

It is the contraction of **else if**

Instruction blocks

- The construction you used with the if instruction is the first example of an instruction block
- An instruction block is a sequence of instructions forming a logical set, which is only executed under certain conditions defined in the header line
- In Python, the blocks always have the same structure: **a header line ending with a colon " : "**, followed by one or more instructions indented **AT THE SAME LEVEL** below this header line

Instruction blocks

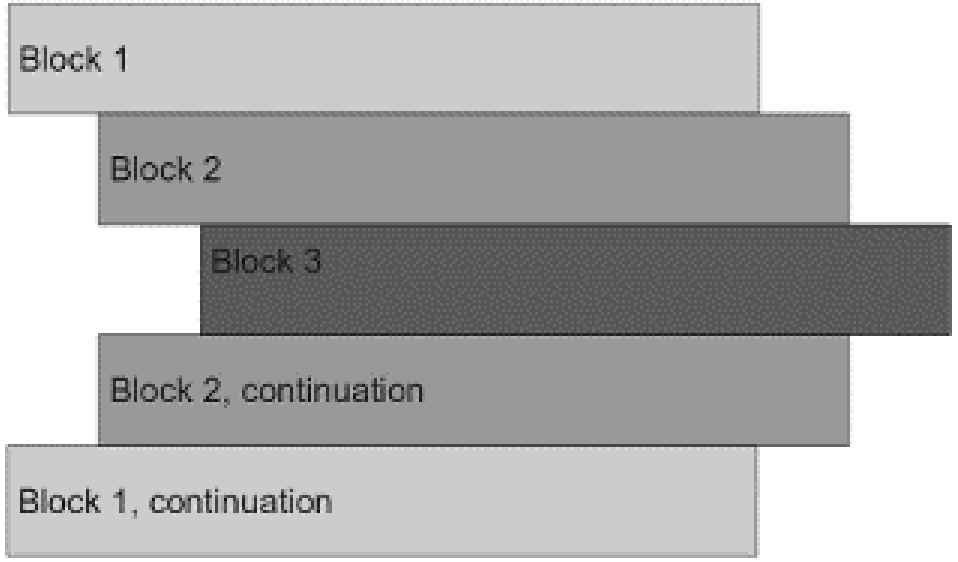
- Example of a block.
  All instructions in the block are indented at the same level.
  This indicates to Python that the instructions belong to this block.

```
Header Line:
    First instruction of the block
    ...
    ...
    ...
    ...
    Last instruction of the block
```

In fact, the indentation can be done with 4 spaces... or 3... or 2 ..
BUT IT HAS TO BE CONSTANT AND WITH NO TABULATION

Instruction blocks

- In Python there is no end of instruction symbol.
- Instruction blocks are always associated with a header line ending with a colon.
- The blocks are delimited by indentation. There is no begin/end or braces to delimit the blocks.
- Note that the code of the outermost block (block 1) cannot itself be excluded from the left margin (it is not embedded in anything).



Loops: While Loop

- Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

- This instruction used at the beginning of a block with a condition tells Python that it must continuously repeat the following block of instructions, as long as the condition is true.

```
while (condition) :
    instruction
    instruction
    instruction
```

With the while statement, Python starts by evaluating the validity of the condition provided in parentheses:

- If the condition is found to be false, then the entire following block is ignored and the program execution resumes at the next block level.
- If the condition is true, then Python executes the entire instruction block constituting the body of the loop

Loops: While Loop

Important!!!

- The variable evaluated in the condition must exist beforehand (it must already have been assigned at least one value).
- If the condition is wrong at first, the loop body is never executed.
- If the condition remains true, then the loop body is repeated indefinitely... until it crashes!
- It must therefore be ensured that the body of the loop contains at least one instruction that changes the value of a variable in the condition evaluated by while so that this condition can become false and the loop can be terminated.

Example 1:

```
a = 0
while (a < 7):
    a = a + 1
    print(a)
```

Example 2:

```
n = 3
while n < 5 :
    print("hello !")
print("I will never be shown ☹")
```

Loops: For Loop

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- The for loop does not require an indexing variable to set beforehand.
- Even strings are iterable objects, they contain a sequence of characters.

Example:

```
for x in "banana":
  print(x)
```

Prints on the screen:

```
b
a
n
a
n
a
```

Loops: For Loop

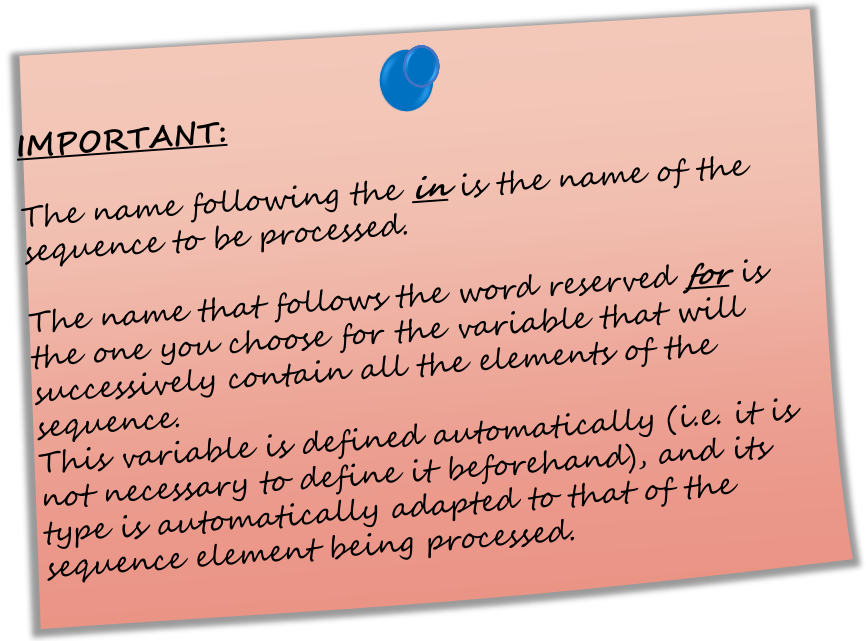
- The for statement therefore makes it possible to write loops, in which the iteration successively deals with all the elements of a given sequence.

- It is possible to create a loop easily with **range**

```
for i in range(0,5):
        print(i)
```

Prints on the screen:

```
0
1
2
3
4
```



in instruction

- The in statement can be used independently of ***for***, to check whether or not a given element is part of a sequence.

Example:

```
n = 5
first =[1, 2, 3, 3, 5, 7, 11, 13, 17]
if (n in premiers):
     print(n, "is on the list of 1st")
```

Forget about the lists for the moment.
We will explain them later on

Exercises

- 3.1 Write a program that, from two integers a and b, displays if a is greater than b.

- 3.2 Write a program that displays whether a variable is positive, negative or null.

- 3.3 Write a program that tests whether an integer is even or odd.
  In the first case, the program will display "a is even" and then, on the next line, the associated mathematical justification. Otherwise the program will display "a is odd".

Exercises

From now on, each of your scripts will contain at least one comment of your choice.

- 4.2 Write a program that displays the first 20 terms of the 7 multiplication table.

- 4.3 Write a program that displays a conversion table of sums of money expressed in euros, in Canadian dollars. The progression of the sums in the table will be "geometric", as in the example below:

  1 euro(s) = 1.45 dollar(s)
  2 euro(s) = 2.90 dollar(s)
  4 euro(s) = 5.80 dollar(s)
  8 euro(s) = 11.60 dollar(s)
  etc. (Stop at 16384 euros.)

Exercises

- 4.4 Write a program that displays a sequence of 12 numbers, where each term is three times the value of the previous term.

- 4.7 Write a program that displays the first 20 terms of the multiplication table by 7, indicating in passing (with an asterisk) those that are multiples of 3.
  Example :
  7 14 21 * 28 35 42 * 49 ...

- 4.8 Write a program that calculates the first 50 terms of the multiplication table by 13, but only displays those that are multiples of 7.

- 4.9 Write a program that displays the following sequence of symbols:
  *
  **
  ***
  ****

- **Data types**

Integer

- Python is virtually capable of processing integers of unlimited size, so that they can fit into the memory of the machine that executes the code.

- Python manages on its own to make the manipulated integers fit into the processor registers on their own.

Float

- For a number to be considered a float number by Python, it is only necessary to include a **.** in its notation or use scientific notation.

Example :

```
1.234 7. 0.001 7e-1
```

Boolean

- A bool variable can have two values:
  True or **False**

- Are considered false:
  - the constants **None** and **False**
  - the different zeros: **0, 0.0**

- The rest is true.

Reminder:

a	not a
False (0)	True (1)
True (1)	False (0)

a	b	a or b
False (0)	False (0)	False (0)
False (0)	True (1)	True (1)
True (1)	False (0)	True (1)
True (1)	True (1)	True (1)

a	b	a and b
False (0)	False (0)	False (0)
False (0)	True (1)	False (0)
True (1)	False (0)	False (0)
True (1)	True (1)	True (1)

String

- Strings are character sequences.
-
- In Python, the elements of a sequence are always indexed (or numbered) starting from zero.

- If you want to determine the number of characters present in a string, you can use the function ***len()***

String

- To extract a character from a string, it is enough to add to the name of the variable that contains this string its index in square brackets :

```
name ='Cedric'
print(name[1], name[3], name[5]) # display e r c
```

- It is often useful to be able to designate the location of a character in relation to the end of the string.
- To do this, it is sufficient to use negative indices. Thus -1 will designate the last character, -2 the second last, etc.

```
print (name[-1], name[-4], name[-6]) # display cdC
```

String

- To extract a string slice we use the syntax :

```
chain[included start index, excluded end index]
```

Example :

```
chain= "Python"
print(chain[0:3]) #prints Pyt
print(chain[:4])  #prints Pyth
print(chain[3:])  #prints hon
```

String

- Operators on Strings:

  There are two operators for character strings:
  - The + operator is used to concatenate strings
  - The * operator is used to repeat a string

Example :

```
chain= “bana”
chain2= chain + “na”
print(chain2*3)    #prints bananabananabanana
```

Exercises

- 5.3 Write a program that converts a temperature expressed initially in degrees Fahrenheit to degrees Celsius, or vice versa.
  The conversion formula is: T F=T C ×1.8+32

- 5.4 Write a program that calculates the interest accumulated each year over 20 years, by capitalizing a sum of 100 euros placed in the bank at a fixed rate of 4.3%.

- 5.6 Write a script that determines whether or not a string contains the character "e".

Exercises

- 5.7 Write a program that counts the number of occurrences of the character "e" in a string.

- 5.8 Write a program that copies a string (into a new variable), inserting asterisks between characters. For example, "gaston" should become "g*a*s*t*o*n".

- 5.9 Write a program that copies a string (into a new variable) by inverting it. For example, "zorglub" will become "bulgroz".

- **Predefined functions**

print() function

- The ***print()*** function allows you to display the value of one or more variables to the screen.
- The default separator (the space) can be replaced by any other character using the ***sep*** argument

```
print("Hello","to","all",sep="*") # prints Hello*to*all
```

- The ***end*** argument allows us to replace the line break:

```
print("omg","omg","omg", end ="!!")
print(“hey“, end ="!!") # prints omgomgomg!!hey!!
```

While:

```
print("omg","omg","omg“) # prints omgomgomg
print(“hey”) # prints hey
```

Each one on a different line

input() function

- The **input()** function causes an interruption in the current program. The user is prompted to enter characters on the keyboard and finish by pressing Enter.

- When this key is pressed, the program continues to run, and the function provides a string of characters corresponding to what the user has entered.

- **This string can then be assigned to any variable, converted**, etc.

ATTENTION:

- **the input() function always returns a string.**

- If you want the user to enter a numerical value, you will therefore have to convert the input value (which will be string type anyway) into a numerical value of the type that you want, via the functions ***int()*** (if you expect an integer) or ***float()*** (if you expect a float one).

Module

- Modules are files that group together sets of functions. They allow you to use code that has already been written.
- It is then a file consisting of Python code.
- A module can define functions, classes and variables. A module can also include runnable code.
- The syntax to use in its program is:

```
from Modulename import *
```

- moduleName allows you to specify the module to import.
- The wildcard (*) imports all the functions of the module.
- We can be more precise and import a specific function.

Example :

```
from math import sinus
```

Exercises

In the following exercise, use the input() function for data entry.

- 6.1 Write a program that converts into meters per second and km/h a speed provided by the user in miles/hour. (Reminder: 1 mile = 1609 meters)

- **User defined functions**

User defined functions

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

- As we previously saw, Python gives you many built-in functions like print(), input() etc. but you can also create your own functions. These functions are called *user-defined functions.*

- The Python syntax for defining a function is:

```
def functionName(parameter list):
    instructionsblock
```

The definition (= code) of the functions must precede their use.

User defined functions

- You can choose any name for the function you create.
- Like the ***if*** and ***while*** instructions, the line containing the ***def*** instruction must end with a colon, which introduces a block of instructions that is **indented**.
- A function is used practically like any other instruction.
- In the body of a program, a function call consists of its name followed by parentheses.
- If necessary, place in these brackets the argument or arguments that you wish to transmit to the function.
- In principle, it will be necessary to provide an argument for each of the specified parameters, although it is possible to define default values for these parameters.

User defined functions

- The return instruction defines the value returned by the function:

```
def cube(w):
    return w*w*w
```

In this case, it is the result of the expression w*w*w.

User defined functions: Function Vs Method

- A function, in the strict sense, must return a value as a result when it ends. While a method may display the result.

- A "true" function can be used to the right of the equal sign in expressions such as `y = sin(a)`
- It is easy to understand that in this expression, the function sin() returns a value (the sinus of argument a) that is directly assigned to the variable y.

```
def cube(w):
    return w*w*w

Res=cube(3)
print (Res) #prints 27
```

A function

```
def cube(w):
    print (w*w*w)

cube(3) #prints 27
```

A method

User defined functions

- Python is a dynamically-typed language, which means that the type of a variable is defined when it is assigned a value.

- This mechanism also works for the parameters of a function. The type of a parameter automatically becomes the same as the type of the argument that was transmitted to the function.

```
def print3times(arg):
    print(arg, arg, arg)

print3times(5)          #prints 5 5 5
print3times('omg')      #prints omg omg omg
print3times(6**2)       #prints 36 36 36
```

Global and local variables

- When we define variables within the body of a function, these variables are only accessible to the function itself.
- These variables are said to be local variables to the function.

- Variables defined outside a function are global variables. Their content is "visible" from within a function, but the function cannot modify it.

```
def mask() :
    p = 20
    print(p, q)
p, q = 15, 38
mask()      # prints 20 38
print(p,q) # prints 15 38
```

Global and local variables

- You may need to define a function that is able to modify a global variable.

- To achieve this result, you will need to use the **global** instruction. This instruction is used to indicate - within the definition of a function - which variables should be treated globally.

```
def increase():
    global a
    a = a+1
    print(a)
a = 15
increase()      #prints 16
increase()      #prints 17
```

Exercises

- 7.2 Define a function lineCar(n, ca) that returns a string of n characters ca.

- 7.5 Define a function maximum(n1,n2,n3) that returns the greatest of 3 numbers n1, n2, n3 provided in arguments.
  For example, the execution of the instruction:
  print(maximum(2,5,4)) #will display 5

- 7.9 Define a function car_count(ca,ch) that returns the number of times the ca character is encountered in the ch character string.
  For example, executing the instruction :
  print(car_count('e','This sentence is an example')) # will display 5

- **Lists**

Lists

- A list is a collection of comma-separated elements, all enclosed in square brackets.

Example :

```
day = ['Monday','Tuesday', 1800, 20.357]
print(day) # prints ['Monday','Tuesday', 1800, 20.357]
```

- day is a list variable.

- Like strings, lists are sequences, i.e. ordered collections of objects.

The various elements that make up a list are always arranged in the same order, and
Each one of them can therefore be accessed individually if you know its index in the list.

- **It should be noted that the numbering of these indexes starts from zero, not from one.**

Lists

- The elements of a list can be replaced by others, as shown below:

```
day[3] =‘July’
print(day) # prints ['Monday','Tuesday', 1800,'July']
```

- The function ***len()*** also applies to lists. It returns the number of elements present in the list:

```
print(len(day)) #prints 4
```

- The ***del()*** function allows you to delete a list item:

```
del(day[2])
print(day) #prints['Monday','Tuesday','July']
```

Lists

- The ***append()*** method adds an element at the end of the list.

```
day.append('Saturday')
print(day) #prints ['Monday','Tuesday','July','Saturday']
```

Exercises

- 5.11 Considering the following lists:
  t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
  t2=['January','February','March','April','May','June','July','August','September','Octo
  ber','November','December']
  Write a small program that creates a new t3 list. This list must contain all the elements of the two lists alternating them, in such a way that each month name is followed by the corresponding number of days:
  ['January',31,'February',28,'March',31, etc...].

- 5.13 Write a program that searches for the biggest item in a given list.
  For example, if applied to the list[32, 5, 12, 8, 3, 3, 75, 2, 15],
  this program should display : the biggest item on this list is 75.

- **Predefined functions**

print() function

- The ***print()*** function allows you to display the value of one or more variables to the screen.
- The default separator (the space) can be replaced by any other character using the ***sep*** argument

```
print("Hello","to","all",sep="*") # prints Hello*to*all
```

- The ***end*** argument allows us to replace the line break:

```
print("omg","omg","omg", end ="!!")
print(“hey“, end ="!!") # prints omgomgomg!!hey!!
```

While:

```
print("omg","omg","omg“) # prints omgomgomg
print(“hey”) # prints hey
```

Each one on a different line

input() function

- The **input()** function causes an interruption in the current program. The user is prompted to enter characters on the keyboard and finish by pressing Enter.

- When this key is pressed, the program continues to run, and the function provides a string of characters corresponding to what the user has entered.

- **This string can then be assigned to any variable, converted**, etc.

ATTENTION:

- **the input() function always returns a string.**

- If you want the user to enter a numerical value, you will therefore have to convert the input value (which will be string type anyway) into a numerical value of the type that you want, via the functions ***int()*** (if you expect an integer) or ***float()*** (if you expect a float one).

Exercises

In the following exercise, use the input() function for data entry.

- 6.1 Write a program that converts into meters per second and km/h a speed provided by the user in miles/hour. (Reminder: 1 mile = 1609 meters)

- **User defined functions**

User defined functions

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

- As we previously saw, Python gives you many built-in functions like print(), input() etc. but you can also create your own functions. These functions are called *user-defined functions.*

- The Python syntax for defining a function is:

```
def functionName(parameter list):
    instructionsblock
```

The definition (= code) of the functions must precede their use.

User defined functions

- You can choose any name for the function you create.
- Like the ***if*** and ***while*** instructions, the line containing the ***def*** instruction must end with a colon, which introduces a block of instructions that is **indented**.
- A function is used practically like any other instruction.
- In the body of a program, a function call consists of its name followed by parentheses.
- If necessary, place in these brackets the argument or arguments that you wish to transmit to the function.
- In principle, it will be necessary to provide an argument for each of the specified parameters, although it is possible to define default values for these parameters.

User defined functions

- The return instruction defines the value returned by the function:

```
def cube(w):
    return w*w*w
```

In this case, it is the result of the expression w*w*w.

User defined functions: Function Vs Method

- A function, in the strict sense, must return a value as a result when it ends. While a method may display the result.

- A "true" function can be used to the right of the equal sign in expressions such as `y = sin(a)`
- It is easy to understand that in this expression, the function sin() returns a value (the sinus of argument a) that is directly assigned to the variable y.

```
def cube(w):
    return w*w*w

Res=cube(3)
print (Res) #prints 27
```

A function

```
def cube(w):
    print (w*w*w)

cube(3) #prints 27
```

A method

User defined functions

- Python is a dynamically-typed language, which means that the type of a variable is defined when it is assigned a value.

- This mechanism also works for the parameters of a function. The type of a parameter automatically becomes the same as the type of the argument that was transmitted to the function.

```
def print3times(arg):
    print(arg, arg, arg)

print3times(5)          #prints 5 5 5
print3times('omg')      #prints omg omg omg
print3times(6**2)       #prints 36 36 36
```

Global and local variables

- When we define variables within the body of a function, these variables are only accessible to the function itself.
- These variables are said to be local variables to the function.

- Variables defined outside a function are global variables. Their content is "visible" from within a function, but the function cannot modify it.

```
def mask() :
    p = 20
    print(p, q)
p, q = 15, 38
mask()      # prints 20 38
print(p,q) # prints 15 38
```

Global and local variables

- You may need to define a function that is able to modify a global variable.

- To achieve this result, you will need to use the **global** instruction. This instruction is used to indicate - within the definition of a function - which variables should be treated globally.

```
def increase():
    global a
    a = a+1
    print(a)
a = 15
increase()      #prints 16
increase()      #prints 17
```

Exercises

- 7.2 Define a function lineCar(n, ca) that returns a string of n characters ca.

- 7.5 Define a function maximum(n1,n2,n3) that returns the greatest of 3 numbers n1, n2, n3 provided in arguments.
  For example, the execution of the instruction:
  print(maximum(2,5,4)) #will display 5

- 7.9 Define a function car_count(ca,ch) that returns the number of times the ca character is encountered in the ch character string.
  For example, executing the instruction :
  print(car_count('e','This sentence is an example')) # will display 5

- 7.10 Define a function indexMax(list) that returns the index of the element with the highest value in the list passed as argument. Usage example:

```
serie = [5, 8, 2, 1, 9, 3, 6, 7]
print(indexMax(serie))  #prints 4
```

- **JSON Format**

JSON Format: Definition

- JSON (JavaScript Object Notation from JavaScript) is a lightweight format for exchanging data between applications. It is easy to read or write for humans.

- It is easily parsed or generated by machines. It is based on a subset of the JavaScript programming language. JSON is a completely language-independent text format, but the conventions it uses will be familiar to any programmer used to languages descending from C, such as: C itself, C++, C#, Java, JavaScript, Perl, Python and many others. These properties make JSON an ideal data exchange language.

- **Reference:** http://www.json.org/jsonfr.html

JSON Format: Usage

- In JSON we write an integer via its value.
  Example: To send the value 12; we simply write 12.

- For a float we also write its value by putting a point for the decimal separator.
  Example : To send the value 23.7 ; we simply write 23.7

- For a character string, the text is written between quotation marks.
  Example: To send the string toto, simply write "toto".

JSON Format: Usage in python

- The JSON library in Python offers 2 methods of data transformation:
  - The first one is used to encode a JSON message into a string : json.dumps (<variable>).
  - The second one is used to decode a JSON message to a Python variable : json.loads (<message>)

JSON Format: Usage in python

Example: A script that saves a variable in JSON in a text file

```
import json

obj = [1, "poire", 2, "pomme"] # Declare a
# variable

print(type(obj))

ch = json.dumps(obj)

print(type(ch))

print(ch)

with open("fichier.txt", "w") as fichier:
# Opens the file (and creates it if it
# doesn't exist) for writing via the 'w'.

    fichier.write(ch) # Write the string to
# the file
```

Example: A script that reads, from a file, a JSON string and transforms it into a python variable

```
import json

with open("fichier.txt", "r") as fichier:
# Open an existing file in reading mode via
# the 'r' key

    ch = fichier.read() # Read the text file
# and save the content in variable

print(type(ch))

print(ch)

js = json.loads(ch) # Transform the string
# into a python value and store it in a
# variable named js

print(type(js))

print(js)
```

JSON Format: Usage in python

Application exercise 1: Test the previous example and add in a comment the output of each print. What do you notice?

Application exercise 2: Do the same thing as exercise 1 without going through files. I.e., 1- Create a list in python, 2- transform it into a JSON variable, 3- display its type, 4- then transform it into a python variable and 5- display the new type.

Application exercise 3: From the usage example you must: 1- Read the file file.txt, 2- Transform its content into a python list, 3- Add the integer "3" and the string "lemons" in the list, 4- Transform back to JSON and save in file.txt, 5- Display each time the type of the variables and their contents.

- for… in… Instruction

for… in… instruction

You can run through a sequence with a loop built around the while statement.

Python offers a loop structure more appropriate than the while loop for sequences, based on the statement pair for ... in ... :

```
nom ="Cléopâtre"
for car in nom:
    print(car + ' *', end =' ')
```

The variable car will contain successively all the characters of the string, from the first to the last.

```
nom = "Cléopâtre"
index =0
while index < len(nom):
    print(nom[index] + '
*', end =' ')
    index = index +1
```

for… in… instruction

The for statement therefore allows you to write loops, in which the iteration successively processes all the elements of a given sequence.

The statement for ... in ... is a new example of a composed statement.

It is possible to create a loop easily with range :

Example:

```
for i in range(0,100):
    print(i)
```

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax: range(start, stop, step)

for… in… instruction

10.6 In an American tale, eight little ducklings are named respectively: Jack, Kack, Lack, Mack, Nack, Oack, Pack and Qack.
Write a small script that generates all these names from the following two strings:prefix = 'JKLMNOP' and suffix = 'ack’
If you use a for ... in ... statement, your script should only have two lines.

10.11 Write a chainList() function that converts a sentence into a list of words.

- **Modules**

Modules

Ex 1: Test the following example using three files created by nano : exmod.py, exmodtest1.py and exmodtest2.py by putting the three files in the same directory.

exmod.py	exmodtest1.py	Exmodtest2.py
def carre(x):	from exmod import *	import exmod
t=x*x	x=carre(5)	x=exmod.carre(5)
return t	print(x)	print(x)

What do you notice?

Ex 2 : Use the modules to repeat exercises 7.5 and 7.10

THANK YOU!