Variables and types:

- **The variables:**

To define a variable in Python:
Syntax: `>>> name_variable = value`
Three topographic rules for the variable's name:

1. Composed of letters, numbers or « _ », no accent
2. The first character is not a number
3. Case sensitive *(ex:* Age ≠ age ≠ AGE*)*

Manipulations of variables:

Python code
```
>>> my_age = 20
>>> my_age = my_age + 2  # "my_age += 2" give the same result.
>>> my_age
22
```

- **The types:**

Python code
```
>>> a = 2                # The variable is an integer: 'int'
>>> b = " Bonjour ! "    # character string: 'str'
>>> c = 2.3              # 'float'
>>> d = complex(3,2)     # 'complex'
d=3+2j
# There exist boolean type too: 'bool'. There are only two values,
true or false.
```

The « Print » function:

Syntax: `>>> print(name_variable)`
`>>> print("character string")`
`>>> print(3.5)`

Python code
```
>>> print(a)
2
>>> print("Hello world !")  # print(b) give the same result.
Hello world !
>>> print("I am {0} and I have {1} cats.".format(my_age,a))
I am 22 and I have 2 cats.
>>> print("a=", a)
a=2
```

Characters form:

Charactere	Meaning/ Usefulness	Python Syntax
#	Allows to insert comments.	`>>> # This is a comments !`

<, >, <=, >=	Comparison operators.	`>>> a=10`
		`>>> b=5`
		`>>> if a<b:`
		`...    instruction1`
		`Execute the instruction1 because the condition1 is true.`

=, +, -, *, /, //(integer division), %(modulo), **(power)	Calculation operators.	`>>> 10 + 2`
		`12`
		`>>> 10/3`
		`3.3333333`
		`>>> 10//3`
		`3`
		`>>> 10%3  # remainder of the division of ten by three.`
		`1`

==, != (différent)	Comparison operators.	`>>> a=10`
		`>>> b=10`
		`>>> if a==b:`
		`...    instruction1`
		`# /!\ "=" assigne a value whereas "==" check an equality.`

\ (antislash)	Allows to begin a new paragraph when an instruction is too long.	

\n	Allows to begin a new paragraph in a character string.	`>>> print("This is\n a comments !")`
		`This is`
		`a comments !`

\t	Allows to insert tabulations in a character string.	

\'	Allows to insert apostrophes.	`>>> 'It\'s very beautiful.'`
		`/!\ The code 'It's very beautiful' would send an error.`

\"	Allows to insert quotation marks.	
", ', ''', """	Introduce a character string.	

Instruction's block:

Syntax:
```
>>> instruction1
...     instruction1.1
...     instruction1.2
...         instruction1.2.1
...     instruction1.3
...
# execution of the instruction's block
```
/!\ The shaping is significant for the understanding of the instruction block .

Operator's form:

Operator	Meaning/Usefulness	Python Syntax

if	Introduces a condition on a variable.	`>>> if condition1:`
		`...   instruction1`

else	Proposes a different instruction if condition1 is false.	`>>> if condition1`
		`...   instruction1`
		`... else:`
		`...   instruction2`

elif	Proposes several instructions if condition1 is false.	`>>> if condition1:`
		`...    instruction1`
		`... elif condition2:`
		`...    instruction2`
		`... elif condition3:`
		`...    instruction3`
		`... else:`
		`...    instruction4`

while	Allows making a loop while a condition is true.	`>>> while condition1:`
		`...    instruction1`

for	Allows making a loop to skim a set.	`>>> for element in set:`
		`...    instruction1`
		`This code do instruction1 for each element in set.`

in	To point out a set.	

and, or	Logical operator which allows checking one or several conditions.	`>>> if condition1 and condition2:`
		`...    instruction1`
		`instruction1 would be executed if condition1 and condition2 are true.`
		`>>> if condition1 or condition2`
		`...    instruction1`
		`instruction1 would be executed if condition1 or condition2 are true.`

not	Logical operator to check the negation of a condition.	

break	Allows to stop a loop, without going on executing the code.	`>>> if condition1:`
		`...    instruction1`
		`... else:`
		`...    break`
		`# The loop stops if condition1 is false.`

continue	Allows to return to the beginning of the loop, without executing the end of the instruction.	`>>> while condition1:`
		`...   if condition1.1:`
		`...      instruction1.1`
		`...      continue`
		`...   if condition2:`
		`# If condition1.1 is true, we execute instruction1.1 and we return to the beginning of the loop without executing the other lines.`

The input() instruction:

Syntax: `>>> message=input() # the user type an ordinary variable which will be saved in the variable « message ».`
<u>comments:</u> this is not available in Flux

The conditional structures:

- **Simple structure in « if »:**

Python Code
```
>>> a=5
>>> if a>0: # the «: » are compulsory
...    print("a is higher than 0.")
... if a<0:
...    print("a is lower than 0.")
...
a is upper than 0.
```

- **Structure in « if, else »:**

Python Code
```
>>> if a>0:
...    print("a is upper than 0.")
... else:   # if the first condition is not true:
...    print("a is lower than 0.")
...
a is upper than 0.
```

- **Complete structure in « if, elif, else »:**

Python Code
```
>>> if a>0:
...    print("a is upper than 0.")
... elif a<0:
...    print("a is lower than 0.")
... else:
...    print("a equal to 0.")
...
a is upper than 0.
```

The Loops:

- **The « while » loop:**

Python Code
```
>>> nb=7
>>> i=0  # Initialization of the variable that will be used in the
loop
>>> while i<10: # the instruction is done while "i<10" is true
...    print( i+1,"*",nb, "=",(i+1)*nb)
...    i+=1  # when the instruction is done, we increment i
...
1*7=7
2*7=14
3*7=21
4*7=28
5*7=35
6*7=42
7*7=49
8*7=56
9*7=63
10*7=70
```

- **The « for » loop:**

Python Code
```
>>> for letter in " Hello! ":  # letter is a variable that stands for
each element of the character string "Hello !"
...   print(letter)
...
H
e
l
l
o
!
```

Other example:

Python Code
```
>>> for letter in "Hello!":
...    if letter in "aeiouyAEIOUI":  # if letter is a  vowel
...        print(letter)
...    else:                         # if letter is a consonant or an
other character
...        print("*")
...
*
e
*
*
o
*
```

Main Notions: classes, methods and objects:

- <u>Objects</u>: data structures, can contain variables and functions (which are called methods). To use the method of an object: "object.method()".
- The classes are data types, models which are used to build an object: it is in a class that we define the methods peculiar to the object.

For instance:

'str' is the character string class.
string="Hello !" is an object of the 'str' class.
lower is a method of the 'str' class which can be applied on all its objects.

The functions:

- **The function creation:**

Use of the « def » command:
Syntax: `>>> def function(param1, param2, …, paramN):`
`...       instructions block`

Python Code
```
>>> def table(nb): # nb stands for the number which the table will be
calculated
...    i=0
...    while i<10:
...        print(i+1,"*",nb,"=",(i+1)*nb)
...        i+=1
...
```
```
>>> table(6)
1*6=6
2*6=12
3*6=18
4*6=24
5*6=30
6*6=36
7*6=42             # etc until 6*10=10
```

Use of the « lambda » command:
Syntax: `>>> lambda arg1,arg2,…,argN: return instruction`

Python Code
```
>>> f= lambda x: x*x  # this function return x squared
>>> f(5)
25
```

Values defaults in a function:
When you define a function, you can initialize a parameter in setting: param=values_in_absentia, thus, if no value is given by the user of the function, the parameter would have the default value.
Unknown numbers of parameters:
When you do not know how many parameter would composed your function, all you have to do is to put a " * " in front of the parameter.

- **Some usefull functions in Python:**

For the 'str' class:

Functions	Usefulness	Python Syntax

upper(), lower(), capitalize(), strip(), center(nb)	To shappe of a character string.	`>>> "chaine".upper()   # contrary to lower()`
		`CHAINE`
		`>>> "chaine".capitalize()`
		`Chaine`
		`>>> "   character string  ".strip`
		`'chaine de caractere'`
		`>>> "chaine".center(30)   # center the string in 30 characters`
		`'            chaine            '`

format()	To call the elements to put them in a character string.	`>>> var1 = value1`
		`>>> var2 = value2`
		`>>> print('"var1 is equal to {0} and var2 to {1}.".format(var1,var2))`
		`var1 is equal to value1 and var2 to value2.`
		`>>> adress = """ {num} {name}`
		`…     {postal_code} {city}`
		`…     """.format(num=5, name="rue des lilas", postal_code=75003, city="Paris")`
		`...`
		`>>> print(adress)`
		`5 rue des lilas`
		`75003 Paris`

len()	To return the length of a string.	`>>> len("Hello!")`
		`6`

exec()	To execute a command shape like a character string.	`>>> exec("if 0==0: print(\"ok\")")`
		`ok`

For all the classes:

Functions	Usefulness	Syntax

type()	Return the type of a variable.	`>>> type("hello")`
		`<class 'str'>`

int(), str(), float()	Change the type of a variable. /!\ The conversion have to be possible.	`>>> chaine="34"`
		`>>> int(chaine)`
		`34`

The exceptions

It allows pointing out the errors which you can meet. We use the commands "try" and "except":
Syntax:
```
try:
    # block to execute
except:
    # block which will be executed in case of
error
```

Python Code
```
>>> try:
...    6/0
... except:
...    print("error")
...
error
```

We can also point out on a special error:

Python Code
```
>>> try:
...    6/0
... except ZeroDivisionError:
...    print("Division by zero impossible.")
...
Division by zero impossible.
```

The « return » instruction:

It allows returning directly a value:
Syntax: >>> `def square(x)`
```
...     return x*x
```
This function gives x squared.

Programs creation:

Steps: 1. Open a basic text editor (ex: bloc note, /!\ Word and WordPad do not work)
2. Header-block:

Python Code
```
-*-coding:CODING -*  # CODING have to be replaced by:
                      #    Latin-1 with Windows
                      #    Utf-8 with Linux
                   # It depends on the coding of your computer

import os # importe the module with the functions and the variables
of Python
```
3. Write the code as if you were in Python
4. Save the project with the extensions « .py »

Use of the program with Python:

- *Program which interact with the user (use of the "input()" instruction for instance):*

Double clic on the file → opening of a Python Windows

- *program with a function (as the function "table()" for instance):*
  1. Save the file in the same folder as Python.
  2. Import the file:

Python Code
```
>>> import my_file
```

Import of a Python index:

Usefullness: Obtain functions usable in the Python commands.
Syntax: >>> `import index`
```
     >>> index.function1(param1, param2, ...,
paramN)
     execution of fonction1
```
or: `>>> from index import function1`
```
     >>> function1(param1, param2, …, paramN)
```
or: `>>> from index import *      # allows to write "function1" without having to write the index`
```
     >>> function1(param1, param2, …, paramN)
```

Python Code
```
>>> import math  # import the math index which contains mathematics
functions: cos, sin, etc.
>>> math.sqrt(16)  # to use the functions of the index, we have to
put the name of the index in front of
4
```

The characters string:

- **The concatenation of the strings:**

Syntax:
```
>>> a="string1"
>>> b="string2"
>>> c=a+b
>>> c
>>> string1string2
```
/!\ If we want to put a space between both strings, we have to write in Python: `>>> c=a+" "+b`
/!\ If we want to add an integer or a float variable in the string, we have to convert them with the function "str()":

Python Code
```
>>> "I am"+" "+str(21)+" "+"years old."
I am 21 years old.
```

- **How to point out on a character string:**

Syntax:
```
>>> string[0]
   # return the first character of the string
>>> chaine[-1]
   # return the last character of the string
>>> chaine[2:5]
   # return the string from the character
number 2 until the character number 5
>>> chaine[2:]
   # return the string from the character
number 2 until the end of the string
```

The lists:

The lists are set of objects. They belong to the 'list' class and allow manipulating several types of objects in the same time.
Comments: We count from 0.

- **List creation:**

Syntax:
```
>>> my_list= []  # creation of an empty list
>>> my_list= list()  # ditto
>>> my_list = [1, 2, 'a', [] ] # this list is
composed of two integers, a character string and an
other empty list
```
We can call a list element by the same method than for the character strings: « my_list[i] » return the i element of the list.

- **Methods of the list class:**

Major comments: The methods of the list class are different from the method seen previously. Indeed, those methods modify the list but post no result whereas the other methods do not change the element but return a result.

Function	Usefullness	Syntax

append()	To add an object at the end of a list.	`>>> my_list=[5,3]`
		`>>> my_list.append(4)`
		`>>> print(my_list)`
		`[5,3,4]`

insert()	To insert an element in a list.	`>>> my_list.insert(2, 3)  # insertion of the interger 3 at the rating 2.`
		`>>> print(my_list)`
		`[5,3,3,4]`

extend()	To concatenate two lists.	`>>> list1=[6,7]`
		`>>> list2=[2,3]`
		`>>> list1.extend(list2)`
		`>>> print(list1)`
		`[6,7,2,3]`
		`>>> list1 + list2`
		`[6,7,2,3,2,3]`

del	To delete an element.	`>>> del my_list[2] # delete the second element of the list`
remove()		`>>> my_list.remove(2) # /!\ delete the first element which is equal to 2 in the list`

pop()	To delete an element and print it.	`>>> my_list=[5,3,3,4]`
		`>>> my_list.pop(1)`
		`5`

count()	To give the number of element which is equal to the parameter.	`>>> my_list.count(3)`
		`2`

index()	To return the position of the parameter in the list.	`>>> my_list.index(5)`
		`1`

reverse()	To return the inverse of the sequence.	`>>> my_list.reverse()`
		`>>> print(my_list)`
		`[4,3,3,5]`

sort()	To sort the list (by alphabetical order and by monotonic order)	`>>> my_list=[1,2,4,3]`
		`>>> my_list.sort()`
		`>>> my_list`
		`[1,2,3,4]`

- **The "range" function:**

Syntax: >>> `range(n)`
```
   # Create a list from 0 to n-1, with n an integer.
```

Python Code
```
>>> for i in range(3):
. . .    print i
. . .
0
1
2
```

- **The list-string conversion:**

Syntax:
```
>>> my_string="Good Morning Everybody"
>>> my_string.split(" ") # the string is cut at
each space and the pieces are put in a list.
["Good", "Morning", "Everybody"]
```
Comments: The "split()" method gives a result but do not change the list.
Syntax:
```
>>> my_list=["Good", "morning", "everybody"]
>>> " ".join(my_list) # this method joins the
elements of the list together to make a
sentence in separating them by a space.
"Good morning everybody"
```

The tuples:

They are not modifiables lists.
Syntax: >>> `empty_tuple = ( )`
```
>>> tuple = (1,)     # /!\ To create a tuple
with one element, we have to put a comma after
because otherwise, Python would think that this is a
variable and remove the parenthesis.
```

The dictionary:

- **Definition of a dictionary:**

Dictonaries are like the lists and the tuples except that we define them with brace "{ }". Each element is defined as a key associated to a value:
Syntax: >>> `my_dictionary = dict()`
```
>>> my_dictionary = {}  # same result: empty
dictionary
>>> dictionary1={key1:value1,key2:value2}
# a dictionary can be defined directly like this too.
```

- **Addition of elements in a dictionary:**

Syntax:
```
>>> my_dictionary[key]=value # add the couple
key/value to the dictionary
>>> my_dictionary
{ key: value }
```
Contrary to the lists, a dictionary is not ordinate. The values are associated to the keys; hence, they do not have an index in the dictionary.

Python Code
```
>>> cupboard = {}
>>> cupboard ["melon"]= 1
>>> cupboard ["manzana"]= 4
>>> cupboard ["pear"]= 3
>>> cupboard   # The elements are not is the order that we have
define them
{'manzana':4, 'melon'=1, 'pear'=3}
```

- **How to delete elements in a dictionary:**

The "del" and "pop" methods:

Python Code
```
>>> del cupboard["manzana"]
>>> cupboard.pop("pear")   # this method return the value associated
to the key "pear" in deleting it.
3
```

- **How to point out the element of a dictionary:**

With the "for .. in" loop:

The Keys:

Python Code
```
>>> for key in cupboard:   # or « for key in cupboard.keys() »
...    print(key)
...
manzana
melon
pear
```

The values:

Python Code
```
>>> for value in cupboard.values():
...    print(value)
1
4
3
```

Reading and writing in a file:

If we have a text file file.txt in the Python folder:
We can read the file:
Syntax:
```
>>> my_file=open("file.txt", "r")
>>> print(my_file.read()) # print the text
write in file.txt
>>> my_file.close()
```
We can write in the file:
/! This operation delete all the text that was written before in the text.
Syntax:
```
>>> my_file=open("file.txt", "w")
>>> my_file.write("First test !")
  12
>>> my_file.close()
```
We can add sentence in the file:
Syntax:
```
>>> my_file=open("file.txt", "a")
>>> my_file.write("New test")
>>> my_file.close()
```

Comments: It is very important to close the file after using it because the "write" method for instance, have a different usefulness depending on if we are in the writing mode or in the adding mode.

The help function:

Syntax: >>> `help(object)`

MEMENTO

Basics of Python for

Flux®

Tools for programming

Before beginning to program in **Python**:

To download Python and install it on your computer:

You have to go to the official Python website **Python.org** and download the version that matches to your computer and processor.

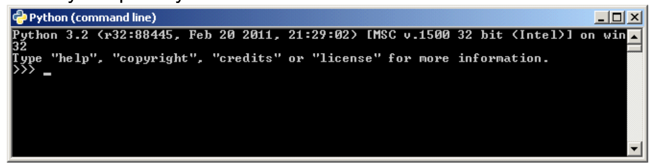
Introduction to Python

Python is an interpreted programming language, that is different from the compiled languages. It automatically compiles the different lines of code each time the user click on "enter".

Nevertheless, it is as well possible to create a python file which will be read by Python and used to do special operations.

A first approach:

When you open Python:



```
Python 3.2 (r32:88445, Feb 20 2011, 21:29:02) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

« >>> » means that you can enter a command.

Help Python website: http://docs.python.org/tutorial

Altair
INNOVATION INTELLIGENCE®