Programming with Python Language and learning algorithm

Serge BOUTER

The content(1)

- The main elements of a computer:
  - Brief description of a computer
  - Memory
  - Data transfer
  - Mass storage
  - Representing numbers
  - ASCII code
- The language Python:
  - environment of development IDLE, text editor, running and debugging
  - Variables and memory,
  - Comparaisons, Boolean expressions and logical operators
  - Strings and how to deal with them
  - Console input/output
  - Flow control: if-else, if-elif-else, for statement, while statement
  - Function, local and global variables, how to make a module
  - Array, tuples and lists, dictionary
- Miscellaneous:
  - Files and folders
  - Numpy Library
  - Displaying charts
  - MySql Database
  - Network, Socket API
  - Developing with Visual Studio Code and creating an virtual environment

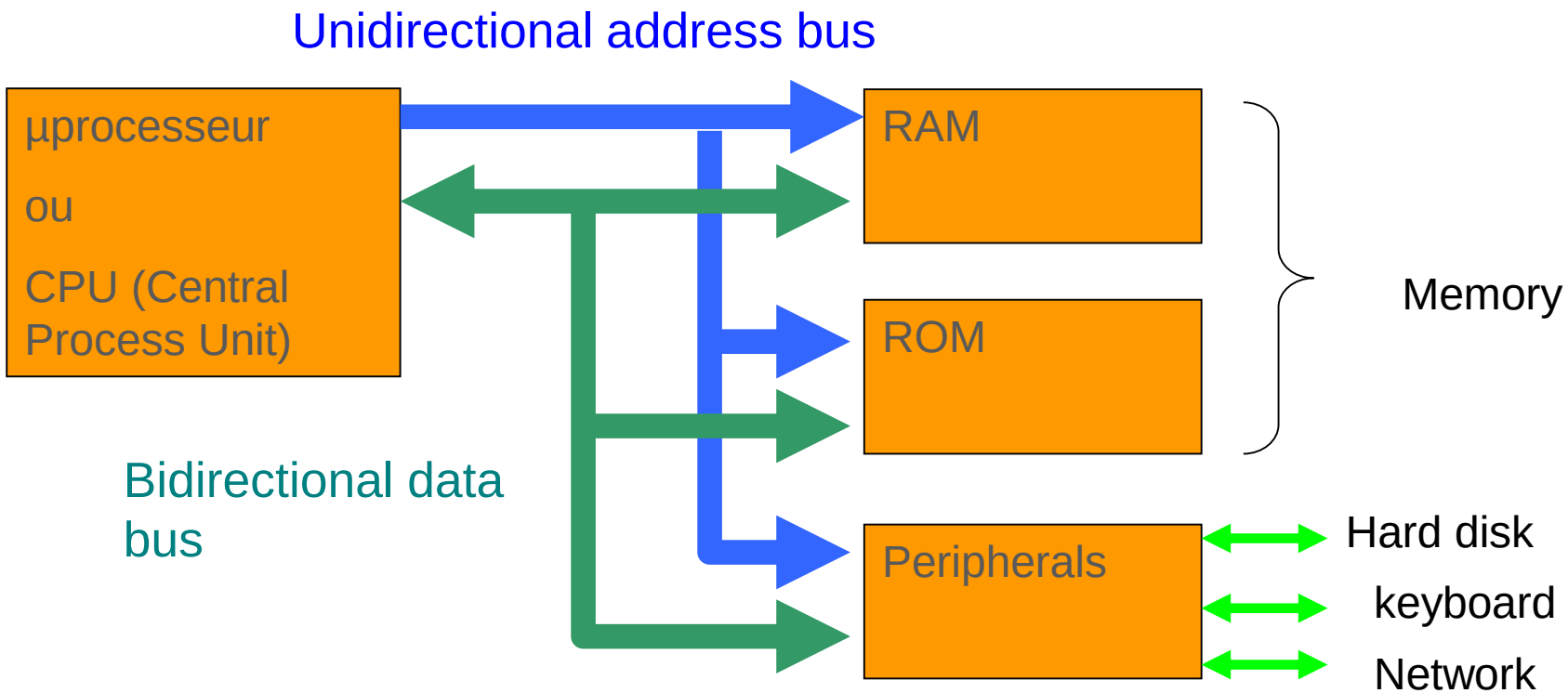
The content(2)

- Assignement exercises:
    - The cash dispenser
    - Checking whether a string of characters matches a number (string, tuple, for and while statement and function)
    - Nought and cross
- Practical Work Exercices on the language
- Object oriented programming:
    - Class
    - Instantiation
    - Inheritance
    - Overloading the operator
    - Class Variable
    - Graphical User Interface Tkinter
- Graphical User Interface: Tkinter
    - A first simple example
    - A second example with a oriented-object programming approach
    - Buttons and callback functions
    - Slider
- Embedded System:Raspberry board

The main elements of a computer

Brief description of a computer
Memory
Data transfer
Mass storage
Representing numbers
ASCII code

Structure of a computer system



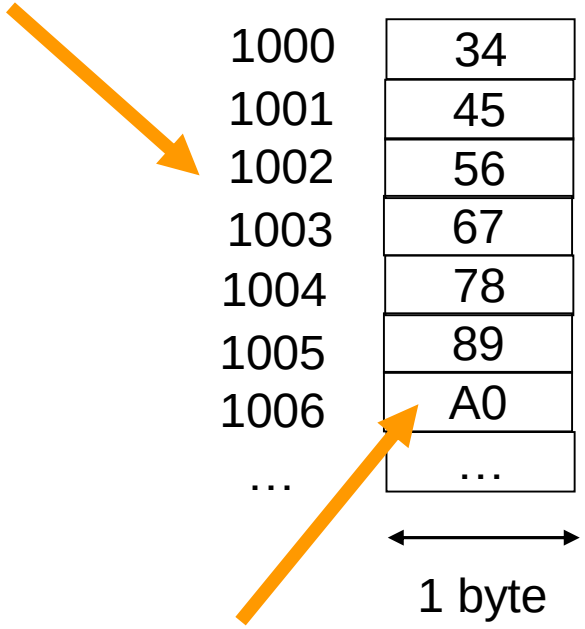
A clock drives the operations inside the CPU

The memory(1)

- The memory space concerns all components capable of storing data: RAM, ROM and peripherals. This space is represented by a series of one-byte memory cells. Each of these memory cells can be selected by the CPU. Thus each memory cells can be **located** by a number called address. The selected memory cells, data transfer between the CPU and the other components is carried out by the data bus.
- RAM: Random Access Memory, can be read or written
- ROM: Read Only Memory, can only be read

The memory(2)

Number of the memory cells or address

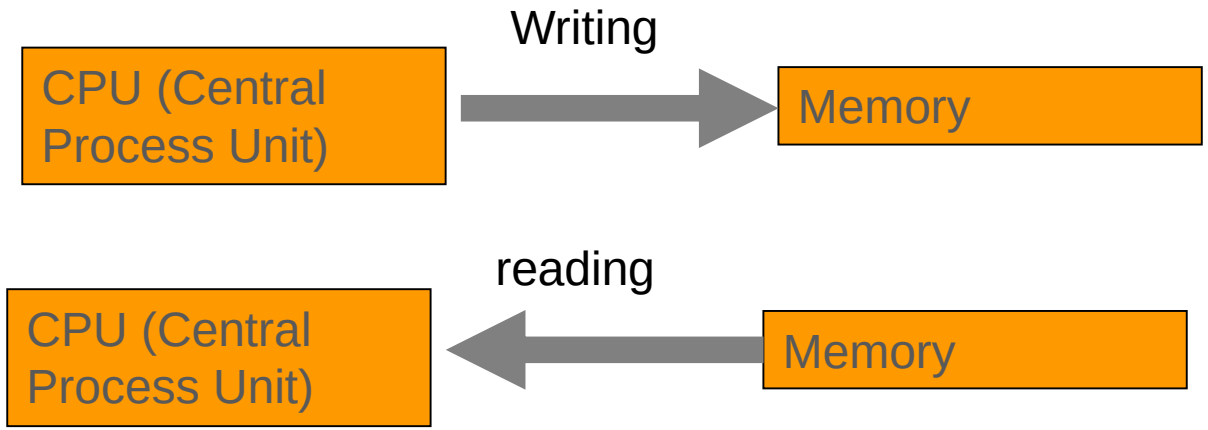


Content of the memory cells in hexadecimal representation

RAM : storing the variable data which can be modified. This part of the memory also stores programmes

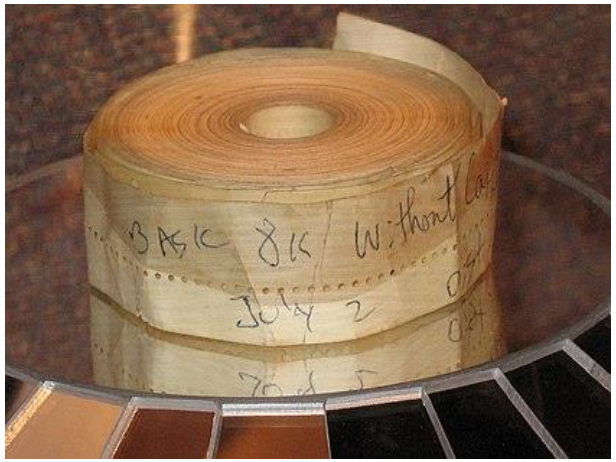
ROM : storing constant data and programmes

Data transfer



The CPU selects with the address of the memory cell on which treatments must be performed

Mass storage



Mass Storage intended for the Basic Language Application. The application is stored on a punched tape which can be read with the computer on the right.

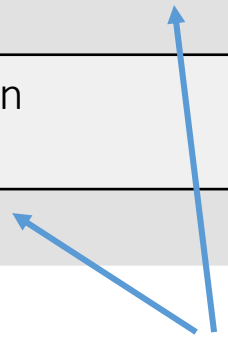
Mass storage is intended for persistent data (infrequently accessed and not likely to be modified) or with more recent device, for virtual memory. Example of mass storage:

- Harddisk drive
- USB stick (Flash memory technology)
- DVD or CD…

Representing numbers

- **The integers:**
- 1 byte -> 2^8 combinations = 256 combinations -> [0,255] ou [-128,127]
- 2 bytes -> 2^{16} combinations = 65 536 combinations -> [0,65 535] ou [-32 768, 32 767]
- **The floating number:**
- The coding of real numbers is based on the IEEE 754 standard. the real simple precision real is coded with 32 bits. This structure consists of the sign, exponent and mantissa fields.

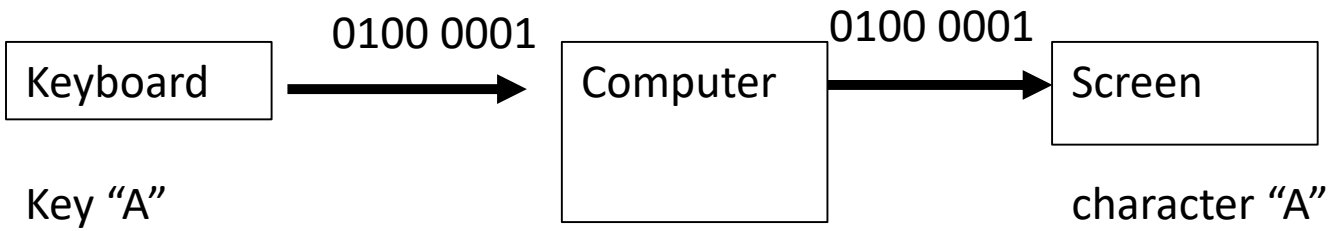
Sign	Exponent	Mantissa	Simple precision
1	8	23	
Sign	Exponent	Mantissa	Double precision
1	11	52	



Because of the sign bit, it exists -0 and +0 even if the others fields are equal to zero

Character code

- Every word is made up of characters. After pressing a key on a keyboard, a number is generated; this number matches the symbol of that key. This is called a character code. A complete collection of characters is a character set.



Content an byte memory cell storing the data matching
$(0100\ 0001)_2$
$(41)_{16}$
$(65)_{10}$

ASCII code (1)
American Standard Code for Information Interchange

code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	NP	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	L	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

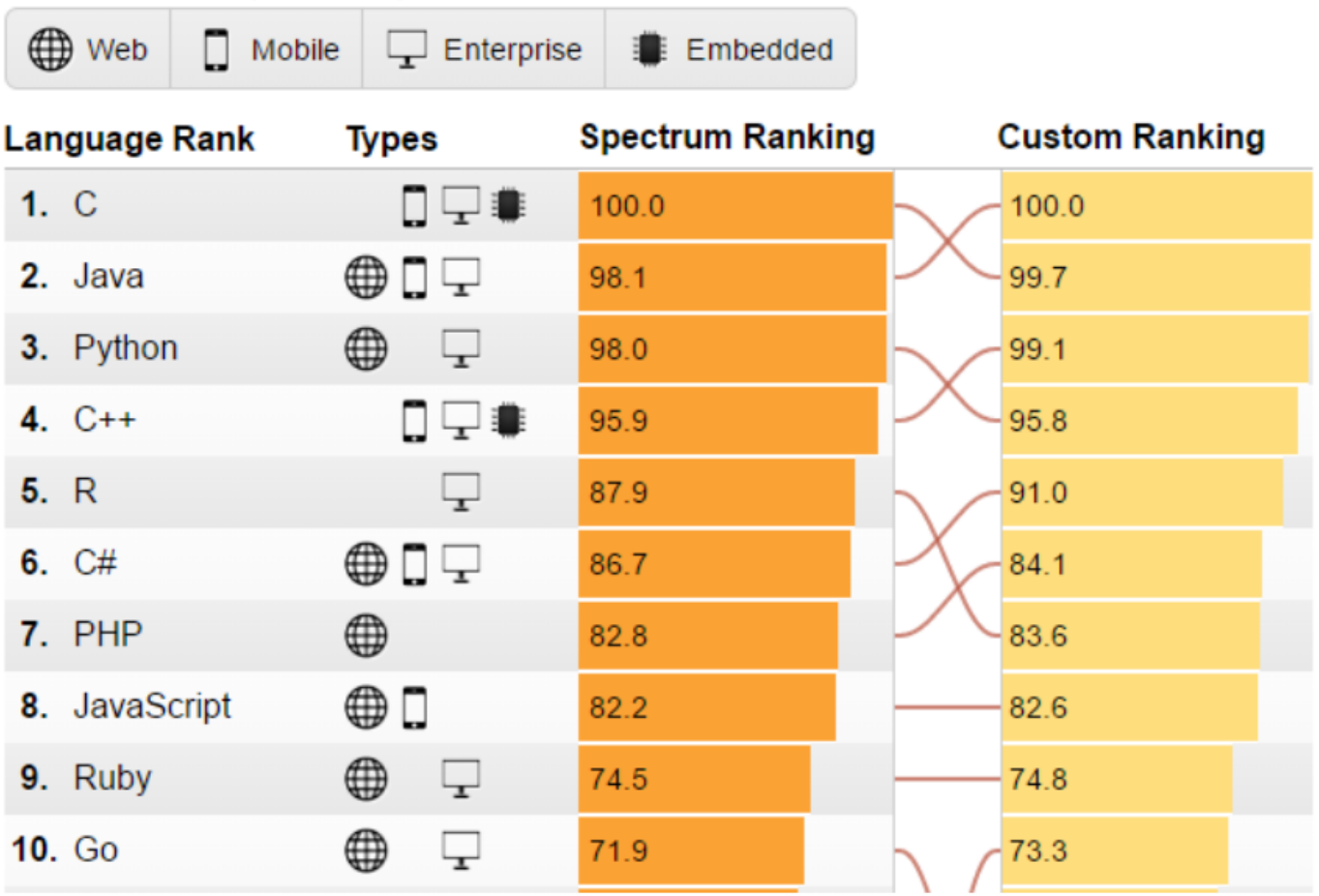
ASCII code (2)
American Standard Code for Information Interchange

code	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	NP	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	SP	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	C
100	d	e	f	g	h	i	j	k	l	M
110	n	o	p	q	r	s	t	u	v	W
120	x	y	z	{	|	}	~	DEL		

The language Python

environment of development IDLE, text editor, running and debugging
Variables and memory,
Comparaisons, Boolean expressions and logical operators
Strings and how to deal with them
Console input/output
Flow control: if-else, if-elif-else, for statement, while statement
Function, local and global variables, how to make a module
Array, tuples and lists, dictionary

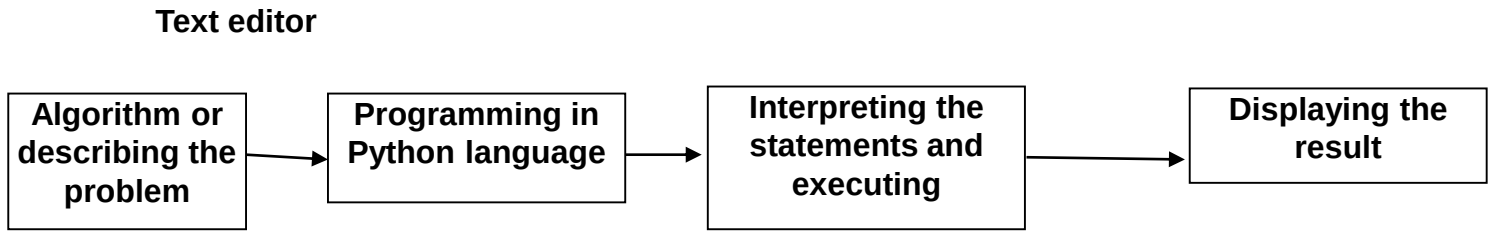
A programming language



IEEE : top 10 des meilleurs langages de programmation de l'année 2016

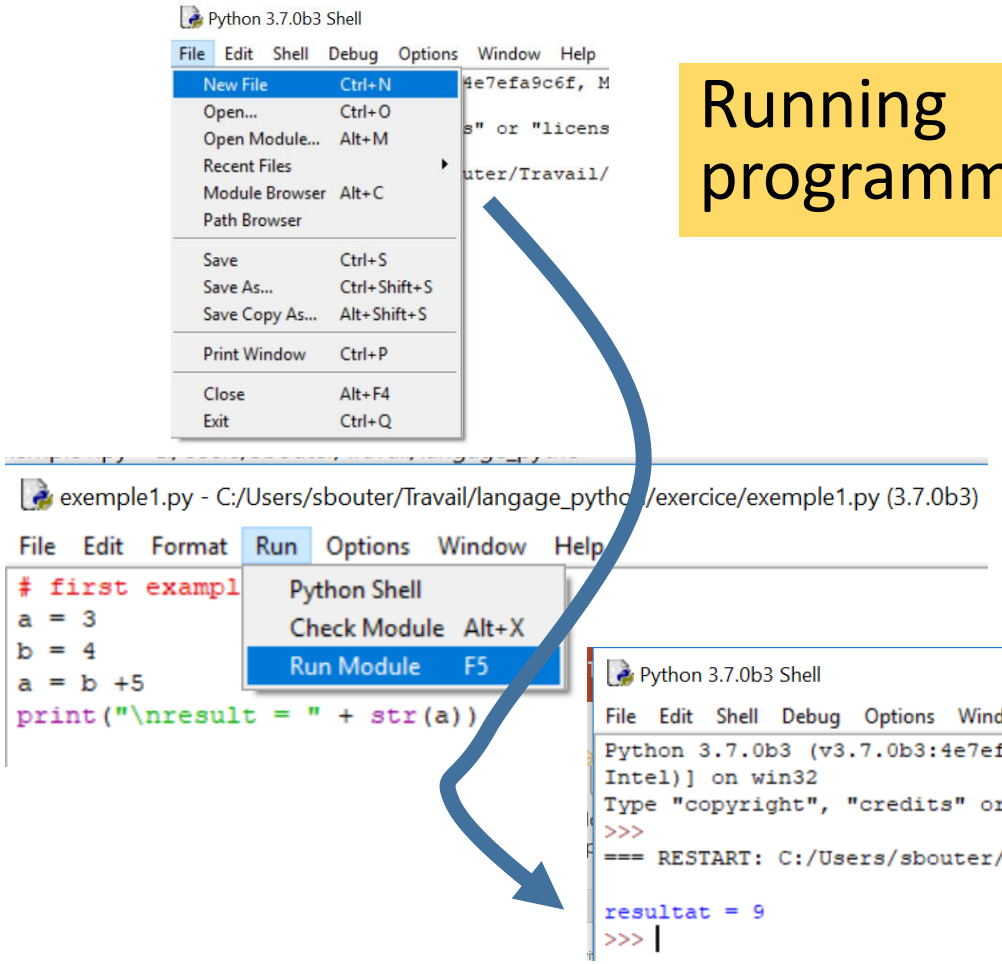
Source developpez.com

From writing the programme to executing it



an interpreter translates source code statements into machine language as the program runs

The environment of development - IDLE



Running programmes

Interpretation and execution of the instruction in line. The prompt “>>>” invites the operator to enter instructions and statements

```
>>> roomNb = 101
>>> id(roomNb)
1405186800
>>> roomNb = 102
>>> id(roomNb)
1405186816
>>> roomNbBooked1 = roomNb
>>> id(roomNbBooked1)
1405186816
>>> roomNbBooked1 = 101
>>> id(roomNbBooked1)
1405186800
>>>
```

The variables and assignement

It is not needed to declare variables which are going to be used and even less to indicate their type. Indeed, a variable can be implicitly declared at the time of its use and typed by assignment.

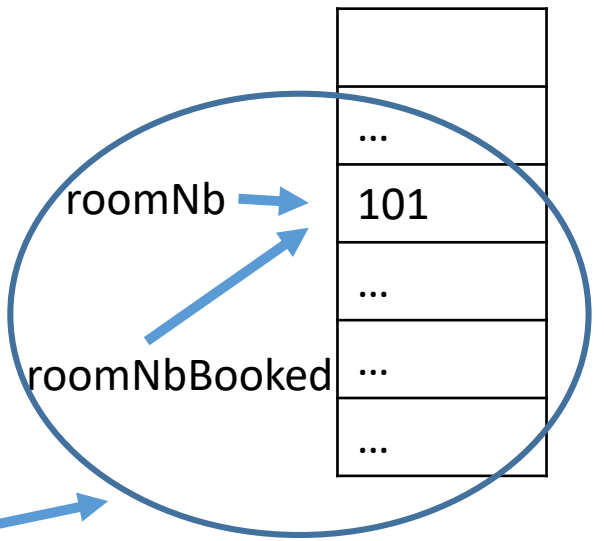
roomNb = 101

This statement makes the value 101 to be stored in memory and the symbolic name “roomNb” references it. Before assigning it, the variable does not exist.

Another kind of assignment can done as follows.

roomNbBooked = roomNb

The symbolic names “roomNb” and “roomNbBooked” reference the same location in the memory. Each variable which has been created is identified by an unique number. This number can be got with the function “id”



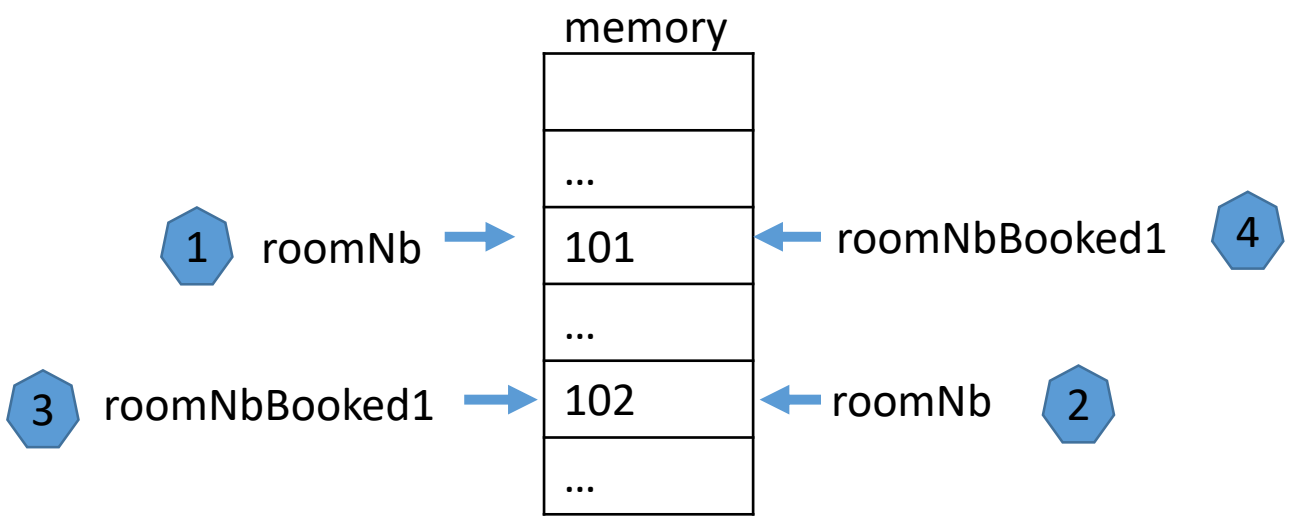

```
>>> roomNb = 101
>>> roomNbBooked = roomNb
>>> id(roomNb)
1455256304
>>> id(roomNbBooked)
1455256304
```

Note: the numbers are said scalar, they cannot be subdivided. The non-scalars have an internal structure that can be accessed, for instance a list of numbers.

The variables and their location in the memory

The variable “roomNb” is assigned the value 101

Let’s what happen to the variable “roomNb” when assigned a new value “102”. Its location in the memory has changed.




```
>>> roomNb = 101
>>> id(roomNb)
1405186800
>>> roomNb = 102
>>> id(roomNb)
1405186816
>>> roomNbBooked1 = roomNb
>>> id(roomNbBooked1)
1405186816
>>> roomNbBooked1 = 101
>>> id(roomNbBooked1)
1405186800
>>>
```

1 2 3 4

Simple type of variables or simple object

- Type integer:

  By default, the presentation of the number is decimal. The hexadecimal presentation is indicated as follows: 0xA458. The “int” type has not a real definite size. Python adapts the size of the storage to the possibility of the machine and the operating system (in other languages as C or Java 32 bits is needed to store an integer).

- Type float:

  Machines today use IEEE-754 floating point arithmetic

- Type bool: true

- How to determine the type of a variable: the type of the variable can be modified along the running of the program. So it can useful to be able to determine the type of a variable. The instruction “type” allows it.

```
>>> type(7)
<class 'int'>
>>> type(7.8)
<class 'float'>
>>> type(roomNb)
<class 'int'>
>>> type('12345')
<class 'str'>
>>> type(FirstText)
<class 'str'>
```

The format of the integers

- Integers in Python 3 are of unlimited size. The limitation comes from the computer, the operating systems and how these latter deal with the memory

```
>>> 0x3FFF
16383
>>> 0b11111111111111
16383
>>> 16383/25
655.32
>>> 16383//25
655
>>> 16383%25
8
>>>
```

```
>>> 2**64
18446744073709551616
>>> 2.0**64
1.8446744073709552e+19
>>> 2**100
1267650600228229401496703205376
>>> 2.0**100
1.2676506002282294e+30
>>>
```

operations with the variables

```
>>> count = 0
>>> id(count)
42298032L
>>> count  = count + 3
>>> id(count)
42297960L
>>> count = count + 5
>>> id(count)
42297840L
>>> count = count + 25
>>> id(count)
42297240L
>>> print(count)
33
>>> 3 + 5 + 25
33
>>>
```

The location of the variable “count” changes at each operation

This sequence of instructions allows the calculations of the sum of three integers. In a programme we do this way to get the sum of several numbers

- - : subtraction
- * : multiplication
- / : division, comment: 5/2 gives 2.5
- // : Euclidean division, comment 5//2 gives 2 (which is the whole number part)
- % : the remainder of the Euclidean division
- ** : power

it exists native functions such as divmod. The function “divmod” gives the quotient and remainder of the Euclidean division

Comparaisons, Boolean expressions and logical operators

- > greater than, >= greater than or equal to,
- < less than, <= less than or equal to
- == equal to

- Exp1 **and** Exp2: the two expressions must be **true** to give **true**
- Exp1 **or** Exp2: one of the two expressions must be **true** to give **true**
- **not** (Exp) Exp: the expression must be **false** to give **true**

- with the logical operator we can get compound Boolean expressions

```
>>> 5 > 7
False
>>> 5 > 4
True
>>> 5 < 89
True
>>> a = 5
>>> b = 34
>>> a >= b
False
>>> a <= b
True
```

```
>>> True and False
False
>>> True and True
True
>>> m = True
>>> n = False
>>> m or n
True
>>> not (m)
False
```

```
>>> not(a >=b)
True
>>> a<=b and m
True
```

```
>>> a>b or m
True
```

The object String of characters

A string: it's a sequence of case sensitive characters which can be letters, digits, special characters (such as /, !, …,#, space) which encloses in single quotes or quotation marks. This object is more complicated than the variables seen before. Indeed the previous variables are said scalar.

- FirstText = 'MyFirstText' # with single quotes
- SecondText = "MySecondText" # with quotation marks
- The character # discards what is written this latter so that it's not been taken into account by the Python runtime.

Some operations:

- Assignement "=" binds the location of a string to another variable name
- Concatenation with the + operator
- Repeating a string a number of times *

```
>>> "python" + " is great"
'python is great'
>>> message1 = "python"
>>> message2 = " is great"
>>> message = message1 + message2
>>> print(message)
python is great
>>> message * 3
'python is greatpython is greatpython is great'
```

How to name the variables

It is important to choose relevant names for the variables. The name must fit the purpose intended to the variable. Some rules have been drawn to make the name legible

- **Camel Case:**

Second and subsequent words are capitalized, to make word boundaries easier to see.

  - Example: numberOfUniversityGraduates

- **Pascal Case:** Identical to Camel Case, except the first word is also capitalized.
  - Example: NumberOfUniversityGraduates
- **Snake Case:** Words are separated by underscores.
  - Example: number_of_university_graduates

Whichever rule you have adopted is important to apply it along your programming

Input and output : the commands “print” and “input”

```
palindromeYear = 2002
stringPalindromeYear = str(palindromeYear)
print ("exemple of palindrome year =",palindromeYear)
print("exemple of palindrome year =" + stringPalindromeYear)
print("exemple of palindrome year =" + " " + stringPalindromeYear) #or
print("exemple of palindrome year = " + stringPalindromeYear)
```

```
exemple of palindrome year = 2002
exemple of palindrome year =2002
exemple of palindrome year = 2002
exemple of palindrome year = 2002
```

The comma inserts a space character

```
number1 = input("Enter a number = ")
a = 9
a = number1 + a
```

The “input” function always returns a string. It is needed to convert this value into a number, an integer or a float

```
Enter a number = 8
Traceback (most recent call last):
  File "C:\Users\sbouter\Travail\langage_python\prepa_cours\print_string.py", line 10, in <module>
    a = number1 + a
TypeError: can only concatenate str (not "int") to str
```

First programme in Python language

```
print("\ninput a number:") # displaying the message “input a number”
number1 = input() # “input” is a pending function. It’s waiting for the end
#of the input to be closed by the “Enter” key pressed. The result of the input is
# stored in a variable called “number1”. At this stage the content is string of numeric characters
# if the input is reliable
number1 = int(number1) # the content of the variable "number1" is converted into an integer
print("\ninput a number:")
number2 = input()
number2 = int(number2)
res = number1 * number2 # performing the multiplication
print("\nresult = " + str(res)) # displaying the message “result=” plus the content
# of the variable “res”. the variable "res" which is an interger is converted into a string of nuneric characters
```

```
input a number:
7

input a number:
9

result = 63
```

Control flow: if and if-else

```
print('hello')
dividend = int(input("Enter a integer dividend = "))
divisor = int(input("Enter a integer divisor = "))
(quotient,remainder) = divmod(dividend,divisor)
if  remainder != 0:
    print(dividend,"is not divisable by",str(divisor))
```

Indented

```
hello
Enter a integer dividend = 8
Enter a integer divisor = 3
8 is not divisable by 3
```

```
hello
Enter a integer dividend = 8
Enter a integer divisor = 4
```

```
hello
Enter a integer dividend = 8
Enter a integer divisor = 4
8 is divisable by 4
```

```
hello
Enter a integer dividend = 8
Enter a integer divisor = 3
8 is not divisable by 3
```

```
print('hello')
dividend = int(input("Enter a integer dividend = "))
divisor = int(input("Enter a integer divisor = "))
(quotient,remainder) = divmod(dividend,divisor)
if  remainder != 0:
    print(dividend,"is not divisable by",str(divisor))
else:
    print(dividend,"is divisable by",str(divisor))
```

Indented

Indented

Let assume that the divisor is different from zero

Control flow: if-elif-else statement

```
print('hello')
dividend = int(input("Enter a integer dividend = "))
divisor = int(input("Enter a integer divisor = "))
(quotient,remainder) = divmod(dividend,divisor)
if  remainder != 0:
    print(dividend,"is not divisable by",str(divisor))
elif dividend == 0:
    print("drop this case... the dividend is equal to",str(dividend))
else:
    print(dividend,"is divisable by",str(divisor))
```

Indented

```
hello
Enter a integer dividend = 8
Enter a integer divisor = 4
8 is divisable by 4
```

```
hello
Enter a integer dividend = 8
Enter a integer divisor = 3
8 is not divisable by 3
```

```
hello
Enter a integer dividend = 0
Enter a integer divisor = 4
drop this case... the dividend is equal to 4
```

Let assume that the divisor is different from zero

Other control flow: the loop structure and while statement and repetitive addition

```
sum = 0
i = 0
while i <10:
    sum = sum + i
    print("intermediate value of sum = ",sum,"when we add",i)
    i = i + 1
print("finally sum = ",sum)
```

Indented

```
intermediate value of sum =  0
intermediate value of sum =  1
intermediate value of sum =  3
intermediate value of sum =  6
intermediate value of sum =  10
intermediate value of sum =  15
intermediate value of sum =  21
intermediate value of sum =  28
intermediate value of sum =  36
intermediate value of sum =  45
finally sum =  45
```

Seen before:
This is the way of performing the addition of several numbers by adding two number several times.
The initial value of “sum” is set to 0

while statement and repetitive multiplication

```
product = 1
i = 1
while i <10:
    product = product * i
    print("intermediate value of product = ",product,"when we multiply by",i)
    i = i + 1
print("finally product = ",product)
```

Indented

```
intermediate value of product =  1 when we multiply by 1
intermediate value of product =  2 when we multiply by 2
intermediate value of product =  6 when we multiply by 3
intermediate value of product =  24 when we multiply by 4
intermediate value of product =  120 when we multiply by 5
intermediate value of product =  720 when we multiply by 6
intermediate value of product =  5040 when we multiply by 7
intermediate value of product =  40320 when we multiply by 8
intermediate value of product =  362880 when we multiply by 9
finally product =  362880
```

Seen before:
This is the way of performing the product of several numbers by multiplying two number several times.
The initial value of “product” is set to 1 and the variable “i” must begin with a value different from 0.

Loop structure and for statement

Indented

```
sum = 0

for i in range(10):
    sum = sum + i
    print("intermediate value of sum = ",sum,"when we add",i)
print("finally sum = ",sum)
```

```
intermediate value of sum =  0 when we add 0
intermediate value of sum =  1 when we add 1
intermediate value of sum =  3 when we add 2
intermediate value of sum =  6 when we add 3
intermediate value of sum =  10 when we add 4
intermediate value of sum =  15 when we add 5
intermediate value of sum =  21 when we add 6
intermediate value of sum =  28 when we add 7
intermediate value of sum =  36 when we add 8
intermediate value of sum =  45 when we add 9
finally sum =  45
```

the range of the value of “i” id from 0 to 9. It is possible to fix the beginning value. Indeed “range” can have up to input parameters

for statement: fixing the beginning value and the step

```
sum = 0

for i in range(4,10):
    sum = sum + i
    print("intermediate value of sum = ",sum,"when we add",i)
print("finally sum = ",sum)
```

setting the beginning value to 4

```
intermediate value of sum =  4 when we add 4
intermediate value of sum =  9 when we add 5
intermediate value of sum =  15 when we add 6
intermediate value of sum =  22 when we add 7
intermediate value of sum =  30 when we add 8
intermediate value of sum =  39 when we add 9
finally sum =  39
```

```
sum = 0

for i in range(4,10,2):
    sum = sum + i
    print("intermediate value of sum = ",sum,"when we add",i)
print("finally sum = ",sum)
```

Setting the step to 2

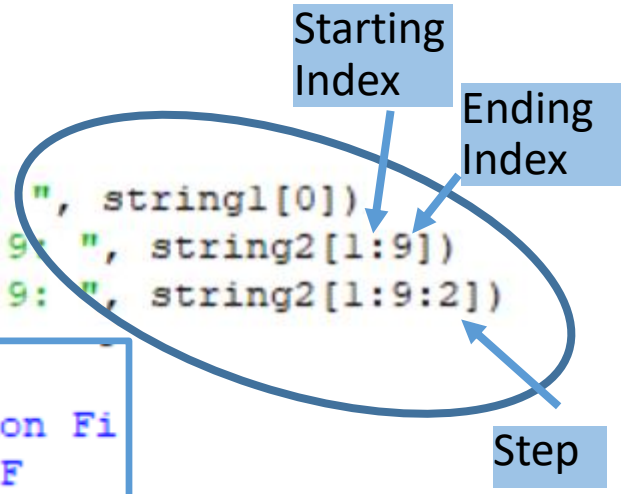
```
intermediate value of sum =  4 when we add 4
intermediate value of sum =  10 when we add 6
intermediate value of sum =  18 when we add 8
finally sum =  18
```

String of characters: how to get access to the characters(1)

How to get access to a particular character or several ? The first character is located with the index equal to zero.

```
string1 = 'Hello World!'
string2 = "Python First Steps"
print ("first character of string1: ", string1[0])
print ("characters from index 1 to 9: ", string2[1:9])
print ("characters from index 1 to 9: ", string2[1:9:2])
```

```
first character of string1:  H
characters from index 1 to 9:  ython Fi
characters from index 1 to 9:  yhnF
```



Strings of characters are immutable

```
print ("third character of string1: ", string1[2])
string1[2] = "h"
```

```
third character of string1:  l
Traceback (most recent call last):
  File "C:\Users\sbouter\Travail\langage_python\prepa_cou
    string1[2] = "h"
TypeError: 'str' object does not support item assignment
```

String of characters: how to get access to the characters(2)

How is the sequence of characters organized?

index		character
0		“0”
1	-2	“1”
2	-1	“2”

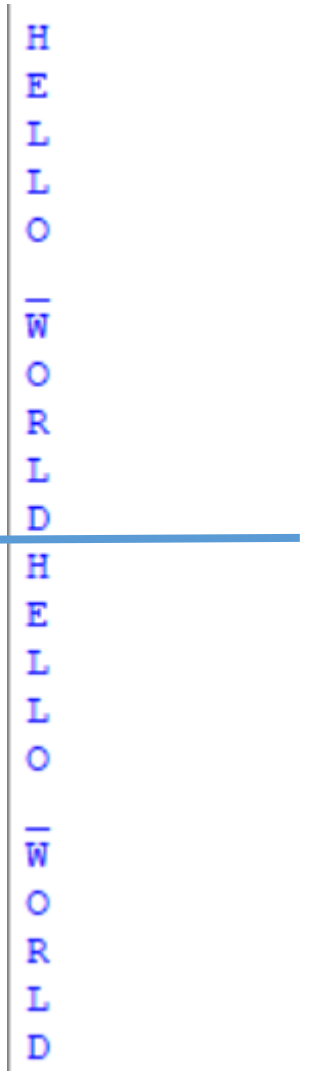
```
string3 = "012"
print(string3[2])
print(string3[1])
print(string3[0])
print(string3[-1])
print(string3[-2])
print(string3[3])
```

```
2
1
0
2
1
Traceback (most recent call last):
  File "C:\Users\sbouter\Travail\langage_python
    print(string3[3])
IndexError: string index out of range
```

Accessing to the last character without knowing the length of the string

String and for statement

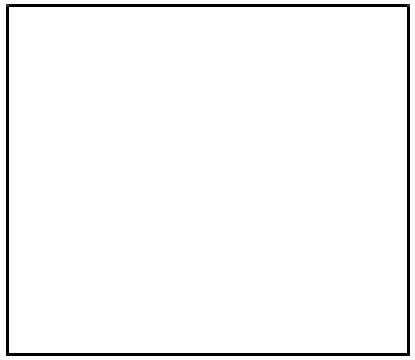
```
stringHello = "HELLO_WORLD"

for charac in stringHello:
    print(charac)

for i in range(len(stringHello)):
    print(stringHello[i])
```



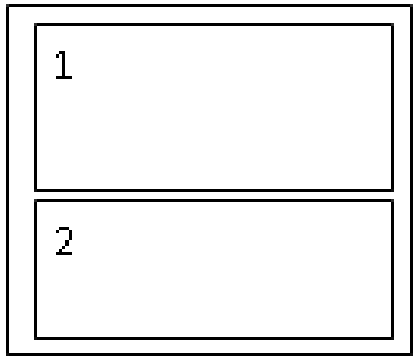
The method len() associated with the string objects gives the length of the strings

How can we organize the writing of a programme

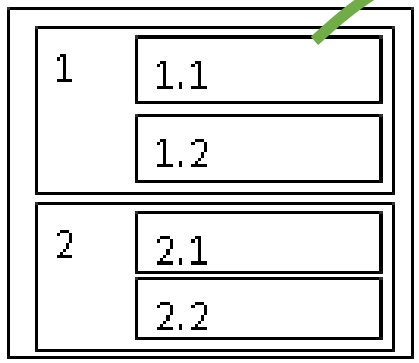
- So far we have just written small programmes, easy to handle
- When it gets too large, it is necessary to split the problem into several sub-problems easier to handle. Each sub-problems can also be split into several sub-problems and so on.
- Each sub-problem can give rise to a function with input and output arguments. So we must increase the level of abstraction to make the programme legible. The abstraction suppresses the details and the functions are viewed as “blackboxes”



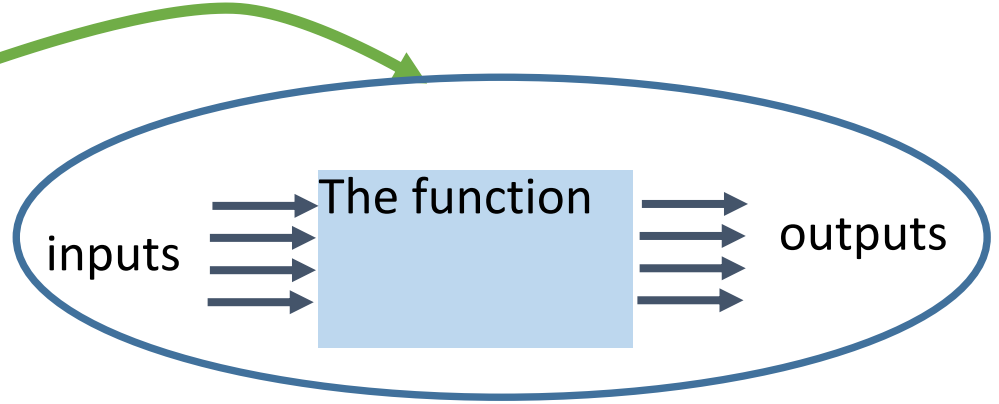
The problem



The sub-problems level 1



The sub-problems level 2



The functions achieve abstraction in writting down their specifications and listing the input and output parameters(docstring are used comment a piece of code)

The functions

The name of the function

The input parameters or formal parameters (have not a specific value). they are also called the arguments of the functions

```
def divisableBy(dividend,divisor):
    ''' this function is intended for testing if a integer, the dividend
        is divisable by another number the divisor. there are 2 input paramaters
        or 2 formal parameters and a result tue return parameter, a boolean.
        '''
    remainder = dividend % divisor
    if  remainder != 0:
        return False
    else:
        return True

                                return (dividend % divisor) == 0

# call the function divisableBy

print(divisableBy(9,3))
print(divisableBy(10,3))
```

Docstring or specification

Can be replaced by this statement

The body of the function

The definition of the function

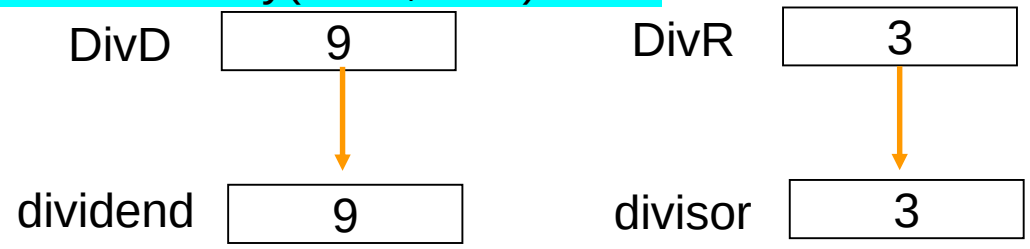
Calling or invoking the function

```
RESTART:
True
False
```

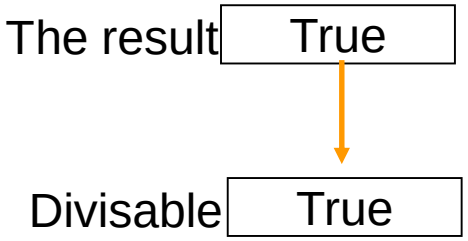
Invoking or calling a function

DivD = 9
DivR =3
Divisable = divisableBy(DivD,DivR)

Passing the argument when calling the function



Treatments



Returning to the calling programme and passing the result

Comment:

The statement "return…" indicates that a value must be returned to the calling programme. In fact, a function, even if return's not given, returns a value "None"

Global and local variables: lifespan and scope(1)

- A global variable can be accessed from anywhere in the file, in any function which is invoked below the definition of this variable. This variable is defined outside of ALL function definitions and his lifespan matches the execution time of the application.
- A local variable can be accessed only in the function in which variable is defined. his lifespan does not exceed the execution time of the function

The variable "remainder" is created. The declaration and the assignment of the local variable occurs inside the function

The variable "remainder" is destroyed at the end of the "body" of the function

```
def divisableBy(dividend,divisor):
    remainder = dividend % divisor
    if  remainder != 0:
        return False
    else:
        return True
```

Global and local variables: lifespan and scope(2)

This assignment is discarded

```
def divisableBy(dividend,divisor):
    # a variable to explain the scope of variable
    #casualVariable = 45678 #a new variable casualVariable but local
    print("inside the function, the value of the ... variable casualVariable= ",casualVariable)
    return (dividend % divisor) == 0

# call the function divisableBy
casualVariable = 1
DivD = 9
DivR =3
DivQ = divisableBy(DivD,DivR)
print(DivQ)
print("outside the function, the value of the global variable casualVariable= ",casualVariable)
```

```
inside the function, the value of the ... variable casualVariable=  1
True
outside the function, the value of the global variable casualVariable=  1
```

Global and local variables: lifespan and scope(3)

This assignment is now effective

```
def divisableBy(dividend,divisor):
    # a variable to explain the scope of variable
    casualVariable = 45678 #a new variable casualVariable but local
    print("inside the function, the value of the ... variable casualVariable= ",casualVariable)
    return (dividend % divisor) == 0

# call the function divisableBy
casualVariable = 1
DivD = 9
DivR =3
DivQ = divisableBy(DivD,DivR)
print(DivQ)
print("outside the function, the value of the global variable casualVariable= ",casualVariable)
```

```
inside the function, the value of the ... variable casualVariable=  45678
True
outside the function, the value of the global variable casualVariable=  1
```

Global and local variables: lifespan and scope(4)

Assignment attempted

```
def divisableBy(dividend,divisor):
    # a variable to explain the scope of variable
    #casualVariable = 45678 #a new variable casualVariable but local
    print("inside the function, the value of the ... variable casualVariable= ",casualVariable)
    casualVariable = 23
    return (dividend % divisor) == 0

# call the function divisableBy
casualVariable = 50
DivD = 9
DivR =3
DivQ = divisableBy(DivD,DivR)
print(DivQ)
print("outside the function, the value of the global variable casualVariable= ",casualVariable)
```

The global variable « casualVariable » is ignored and as a result the programme has failed to continue

```
  File "C:\Users\sbouter\Documents\IUT\langage_python\prepa_cours\functions Divisable.py", line 26, in divisableBy
    print("inside the function, the value of the ... variable casualVariable= ",casualVariable)
UnboundLocalError: local variable 'casualVariable' referenced before assignment
```

Global and local variables: lifespan and scope(5)

Use the keyword « globale to inform « python » it is a global variable

```
def divisableBy(dividend,divisor):
    # a variable to explain the scope of variable
    #casualVariable = 45678 #a new variable casualVariable but local
    global casualVariable
    print("inside the function, the value of the ... variable casualVariable= ",casualVariable)

    casualVariable = 23
    return (dividend % divisor) == 0

# call the function divisableBy
casualVariable = 50
DivD = 9
DivR =3
DivQ = divisableBy(DivD,DivR)
print(DivQ)
print("outside the function, the value of the global variable casualVariable= ",casualVariable)
```

```
inside the function, the value of the ... variable casualVariable=  50
True
outside the function, the value of the global variable casualVariable=  23
```

Integrating the functions in a module

It’s more convenient to store the functions dealing with the same kind of data in a one and same module (operation on measurments… for instance)

The “import” statement allows the programme to retrieve all the functions stored in the module

```
def divisableBy(dividend,divisor):
    ''' this function is intended for testing if a integer, the dividend
        is divisable by another number the divisor. there are 2 input paramaters
        or 2 formal parameters and a result tue return parameter, a boolean.
        '''
    return (dividend % divisor) == 0
```

moduleFunctionDivisable.py

```
import os
print(os.getcwd())
os.chdir('C:\\Users\\sbouter\\Travail\\langage_python\\prepa_cours')
print(os.getcwd())
import moduleFunctionDivisable
print(moduleFunctionDivisable.divisableBy(45,7))
```

Print the current folder

Change the folder

Import the content(function and variable) of the module

Calling the function

To call the function we must refer to the module.

A new data type, the tuples

- A tuple is sequence or a collection of elements of any type (integer, float, Boolean, string)
- Each element is located by an index. The index of the first element is equal to 0
- An empty tuple is obtained by the following statement: emptyTuple = ()

```
tuple1 = ('football', 'rugby', 1998, 2000)
tuple2 = (1, 2, 3, 4, 5, 6)
print ("tuple1[0]: ", tuple1[0])
print ("tuple2[0:3]: ", tuple2[0:3])
print ("tuple2[2:3]: ", tuple2[2:3])
#tuple2[0] = 5
```

```
tuple1[0]:  football
tuple2[0:3]:  (1, 2, 3)
tuple2[2:3]:  (3,)
```

The comma, It's not an error. It indicates we deal with a tuple and not with an integer

```
>>> emptyTuple =()
>>> emptyTuple = emptyTuple + (1)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    emptyTuple = emptyTuple + (1)
TypeError: can only concatenate tuple (not "int") to tuple
>>> emptyTuple = emptyTuple + (1,)
>>> emptyTuple
(1,)
```

```
tuple1 = ('football', 'rugby', 1998, 2000)
tuple2 = (1, 2, 3, 4, 5, 6)
print ("tuple1[0]: ", tuple1[0])
print ("tuple2[0:3]: ", tuple2[0:3])
tuple2[0] = 5
```

```
tuple1[0]:  football
tuple2[0:3]:  (1, 2, 3)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-eb4c16f802ec> in <module>
      3 print ("tuple1[0]: ", tuple1[0])
      4 print ("tuple2[0:3]: ", tuple2[0:3])
----> 5 tuple2[0] = 5

TypeError: 'tuple' object does not support item assignment
```

Immutable like the string

How to deal with the tuples

```
tuple3 = tuple1 + tuple2
print(tuple3)
print (len(tuple3))
tuple4 = tuple3
print(id(tuple3))
print(id(tuple4))
```

Concatenating two tuples

Getting its length or the number of elements

Two names reference the same object

```
('football', 'rugby', 1998, 2000, 1, 2, 3, 4, 5, 6)
10
2304098758856
2304098758856
```

- The native function “divmod” return a tuple : (q,r) = divmod(45,7)

Another new data type, the Lists

- The lists look like tuples. The main difference is the mutability of the Lists
- An empty list is obtained by the following statement: emptyList = []
- The elements of the list can be elements of any type: int, float, Boolean, string.
- nested lists is possible [1,2,[12,23,56],6]

```
mylist = [0,1,2,3]
mylist.append(40)
mylist.append(50)
mylist.append(60)
print(mylist)
print(mylist[0])
print(mylist[3])
print(mylist[6])
```

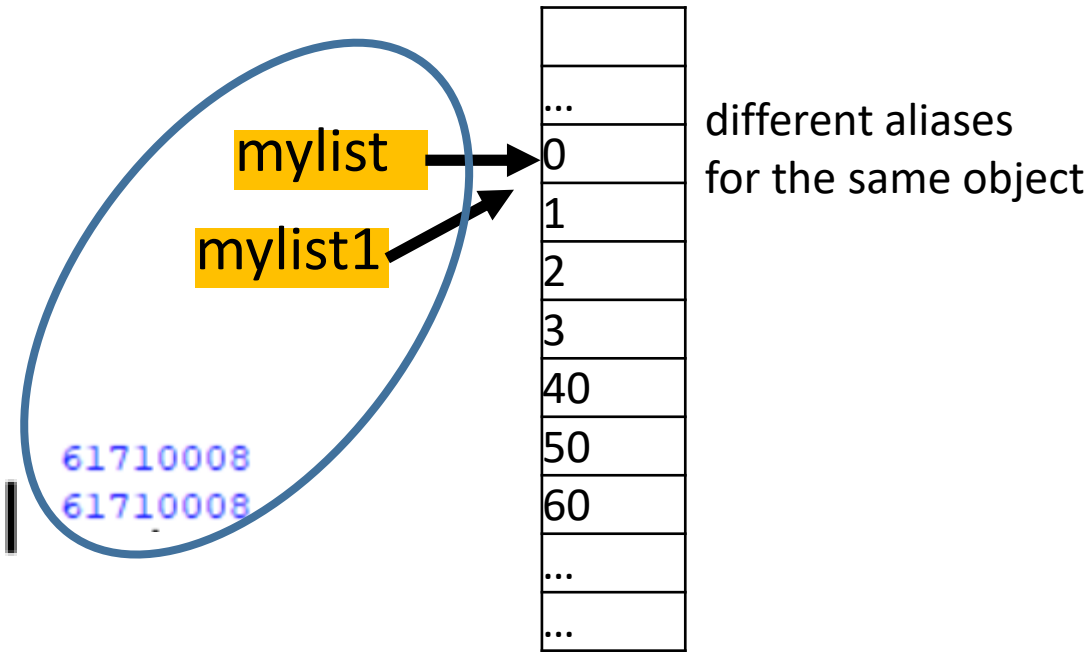
Adding a element at the end of the list

To call the function we must refer to the object

```
[0, 1, 2, 3, 40, 50, 60]
0
3
60
```

```
print(id(mylist))
mylist1 = mylist
print(id(mylist1))
```

```
61710008
61710008
```



Dealing with the Lists

```
myList = [0,1,3,5]
for i in range(len(myList)):
    myList[i] += 3
    #myList[i] = myList[i] + 3

print(myList)
```

```
[3, 4, 6, 8]
```

Each element of the list has been modified; the list has kept the same location in the memory

Suppressing some elements of the list

```
myList1= [0, 1, 3, 5]
myList1= [0, 1, 3, 5, 10, 11, 15]
myList1= [0, 1, 3, 5, 10, 11, 15, 5, 20]
myList1[1:4]= [1, 3, 5]
myList1= [0, 1, 5, 10, 11, 15, 5, 20]
myList1= [0, 1, 10, 11, 15, 5, 20]
```

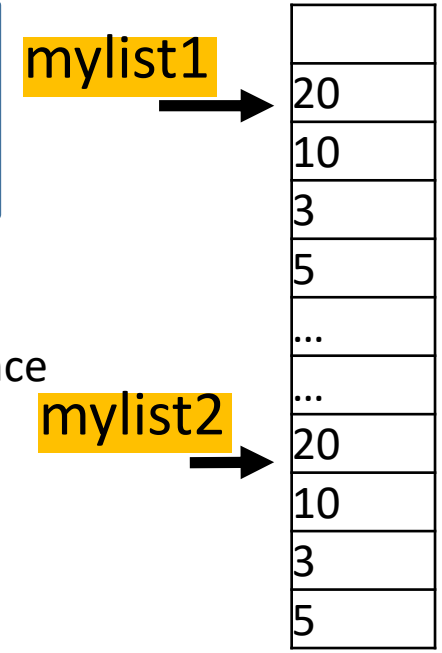
```
myList1 = [0,1,3,5]
print("myList1=", myList1)
myList2 = [10,11,15]
myList1 = myList1 + myList2
print("myList1=", myList1)
myList1.extend([5,20])
# add another list at the end of the first one
print("myList1=", myList1)
print("myList1[1:4]=", myList1[1:4])
del(myList1[2])
# suppress the 2 index element
print("myList1=", myList1)
myList1.remove(5)
# suppress the first element equal to 5
print("myList1=", myList1)
```

Others functions associated with the lists

```
myList1 = [20,10,3,5]
myList2 = myList1[:]
print("myList1=", myList1)
print("myList2=", myList2)
print("id of myList1=", id(myList1))
print("id of myList2=", id(myList2))
```

```
myList1= [20, 10, 3, 5]
myList2= [20, 10, 3, 5]
id of myList1= 54906968
id of myList2= 54907248
```

Creating a second instance

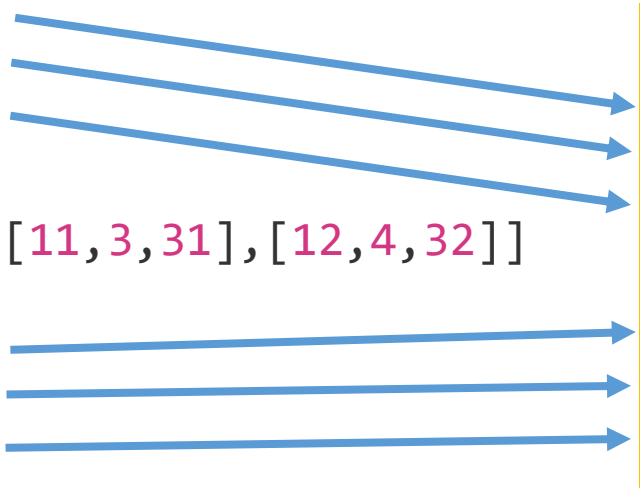


other useful functions:

- sort
- reverse
- min
- max
- split

Multi-Dimension List

```
myList1=
[[10,2],[11,3],[12,4],[14,5],[15,6],[16,7],[17,
8]]
print(myList1)
print(myList1[1][0])
print(myList1[2][1])
print(myList1[6][0])

myList2= [[10,2,30],[11,3,31],[12,4,32]]
print(myList2)
print(myList2[0][0])
print(myList2[1][1])
print(myList2[2][2])
```



```
[[10, 2], [11, 3], [12, 4], [14, 5], [15, 6], [16, 7], [17, 8]]
11
4
17
[[10, 2, 30], [11, 3, 31], [12, 4, 32]]
10
3
32
```

Example of programme using a list

```
print('hello')
b = 3
countDivB = 0
countNotDivB = 0
series =[6,9,3,5,78,45,55,23,27]
for integer in series:
    (q,r) = divmod(integer,b)
    if  r == 0:
        countDivB = countDivB + 1

    else:
        countNotDivB = countNotDivB + 1
print('divisable by B',str(countDivB))
print('not divivable by B',str(countNotDivB))
```

This programme is intended to count the numbers of the list which are divisible by “b”

Dictionary

Each element of a list can be accessible with an index. It's just a number. In this respect, this way of locating can be useful but not relevent as for the content of this element. The dictionary adds a kind of identifier, called a key, allowing the data associated with this key to be accessed. And this way turns out to be more relevant when treating data.

Likewise an index, a key must be unique. you cannot find 2 occurrences of a same key

A list

0	val0
1	Val1
2	Val2
3	val3

A Dictionary

"Thorn"	val0
"Kendall"	Val1
"Kaplan"	Val2
…	val3

The values are stored in a list

The values or data

```
studentDictionary =
{"Thorn":[20,"Strasbourg"],"Kendall":[19,"Heidelberg"],"Kaplan":[21,"Bergamo"]}
```

The keys, immutable

Dictionary: adding and deleting entries

```
#deleting an entry

del(studentDictionary["Thorn"])
print("\nthe content of the dictionay after deleting an entry")
print(studentDictionary)
```

```
# adding a new entry

studentDictionary["Ferguson"] = [18,"Wilhelmshaven"]
print("\nthe content of the dictionary after adding a new entry")
print(studentDictionary)
```

```
the content of the dictionay
{'Thorn': [20, 'Strasbourg'], 'Kendall': [19, 'Heidelberg'], 'Kaplan': [21, 'Bergamo']}

the content of the dictionay after adding a new entry
{'Thorn': [20, 'Strasbourg'], 'Kendall': [19, 'Heidelberg'], 'Kaplan': [21, 'Bergamo'], 'Ferguson': [18, 'Wilhelmshaven']}

the content of the dictionay after deleting an entry
{'Kendall': [19, 'Heidelberg'], 'Kaplan': [21, 'Bergamo'], 'Ferguson': [18, 'Wilhelmshaven']}
```

Dealing with a Dictionary

- An empty dictionary: studentDictionary = {}
- The “clear” method deletes all the elements of the dictionary
- studentDictionary.keys() returns all the keys of the dictionary
- studentDictionary.values() returns all the values of the dictionary
- studentDictionary.get(“the key”) returns the value associated with a specified key
- …

An Exemple:

```
studentDictionary = {"Thorn":[20,"Strasbourg"],"Kendall":[19,"Heidelberg"],"Kaplan":[21,"Bergamo"]}
for myKeys in studentDictionary:

    if (studentDictionary[myKeys][0]>19):
        print(myKeys,studentDictionary[myKeys][0],studentDictionary[myKeys][1])
```

The values or data you want to be printed

```
== RESTART: C:\Users\sbouter\Travail\langage_python\prepa_cours\dictionary.py ==
Thorn 20 Strasbourg
Kaplan 21 Bergamo
>>>
```

Testing and debugging

Defensive programming:

- Modularise the programme
- Draw up specifications for the functions
- Check conditions on input/output

Testing and Validation:

- Compare inputs/outputs with the specifications.
- Spot what is not working
- Rewrite the programmes

Debugging:

- Study events leading to an error
- Figure out why it is not working
- Find how to fix the programme

A way of checking the boundaries:

- Exceptions
- Assertions

Beginning by the simplest part

“One of Linus's earlier projects was a program that would switch between printing AAAA and BBBB. This later evolved to Linux” – quotation of Larry Greenfield

The exceptions and « try and except » statement

An exception corresponds to the occurrence of an error while the programme is running and give rise to halting the programme. There is a way to handle the exception without stopping abruptly the programme, the “try and except” statement

```
try:
    a = int(input("Tell me one number: "))
    b = int(input("Tell me another number: "))
    print("a/b = ", a/b)
except:
    print("Bug in user input.")
```

Divisor equal to zero

```
Enter a number: 4
Enter another number: 0
Bug in user input.
```

```
Enter a number: 4
Enter another number: A
Bug in user input.
```

Not a number

The origin of the error is not identified

Finding the origin of the exceptions

It is possible the find the origin of the exceptions in specifying it

May raise a “ValueError” exception

May raise a “ZeroDivisionError” exception

```
try:
    a = int(input("Enter a number: "))
    b = int(input("Enter another number: "))
    print("a/b = ", a/b)
except ValueError:
    print("it's not a number")
except ZeroDivisionError:
    print("Can't divide by zero")
except:
    print("bug in the programme")
```

The origin of the error is identified

Divisor equal to zero

```
Enter a number: 8
Enter another number: 0
Can't divide by zero
```

Not a number

```
Enter a number: A
it's not a number
```

A more elaborate way of programming can prevent the exception from being raised. In the example above, the value “a” can be tested

Exceptions and Handlers, Raising exceptions

Exceptions and Handlers

- SyntaxError: Python can't parse program
- NameError: local or global name not found
- AttributeError: attribute reference fails
- TypeError: operand doesn't have correct type
- ValueError: operand type okay, but value is illegal
- IOError: e.g. file not found

```
age = input("Enter your age: ")
try:
    age = int(age) # On tente de convertir l'année
    if age <= 0:
        raise ValueError("the age must be greater than 0")
except ValueError:
    print(" invalid value ")
else:
    print ("everything is fine")
```

May raise a "ValueError" exception

```
Enter your age: -5
 invalid value
```

```
Enter your age: 5
everything is fine
```

The "raise" statement make it possible to draw up customised exceptions

Assertions

The assertions check the conditions are well fulfilled and so prevent the error from propagating along the running programme

```
Enter your age: -4
Traceback (most recent call last):
  File "C:\Users\sbouter\Travail\langage_python\prepa_cours\exception.py", line
65, in <module>
    assert age >= 0, "age cannot be negative"
AssertionError: age cannot be negative
```

```
age = int(input("Enter your age: "))
assert age >= 0, "age cannot be negative"
```

```
Traceback (most recent call last):
  File "C:\Users\sbouter\Travail\langage_python\prepa_cours\exception.py",
    print(factorial(5.5))
  File "C:\Users\sbouter\Travail\langage_python\prepa_cours\exception.py",
    assert type(n) == int
AssertionError
```

```
def factorial(n):
    assert type(n) == int
    res = 1
    for i in range(1,n+1):
        res = res * i
    return res
print(factorial(5.5))
```

Miscelleaneous

Files and folders

Numpy Library

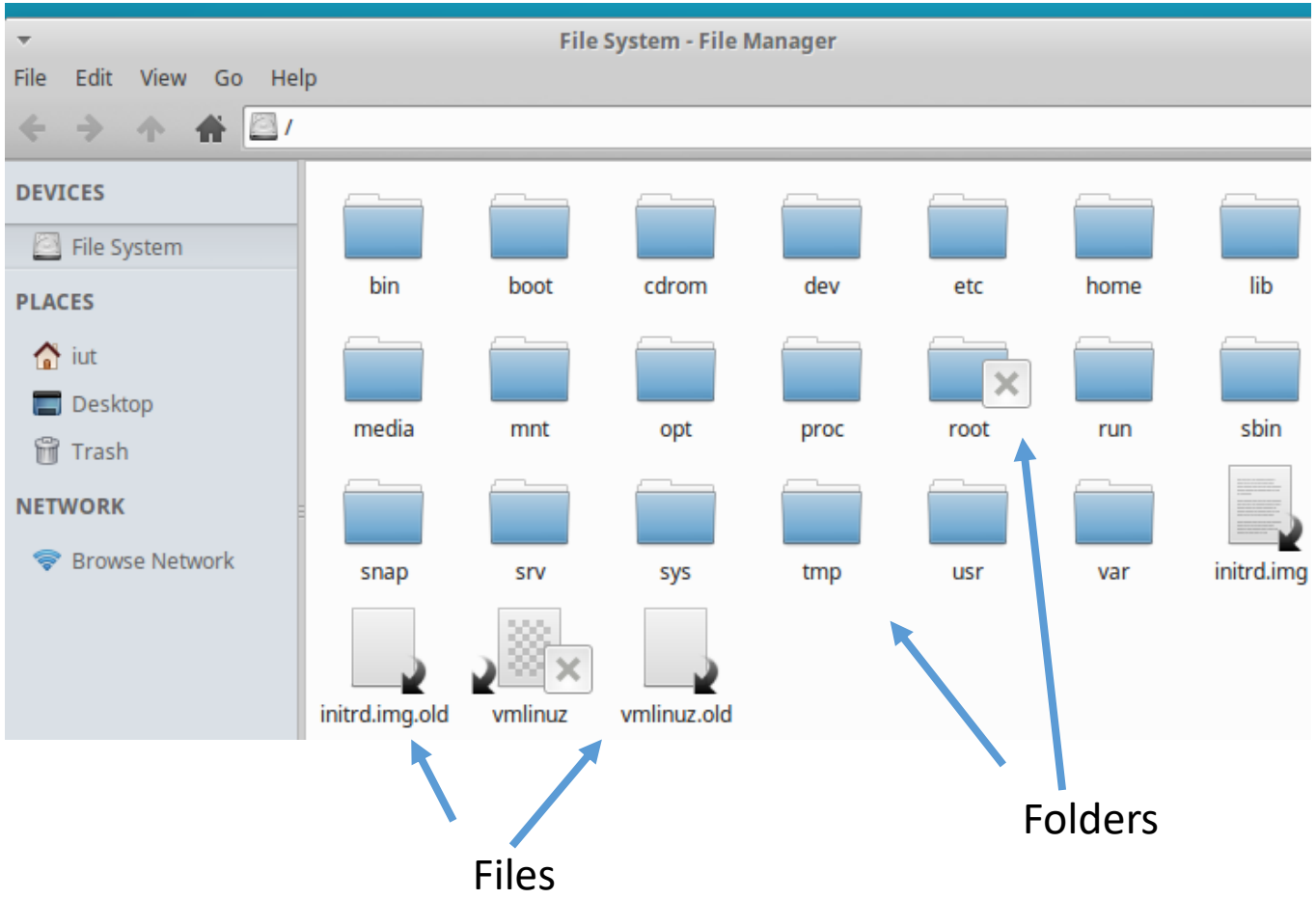
Displaying charts

MySql Database

Network, Socket API

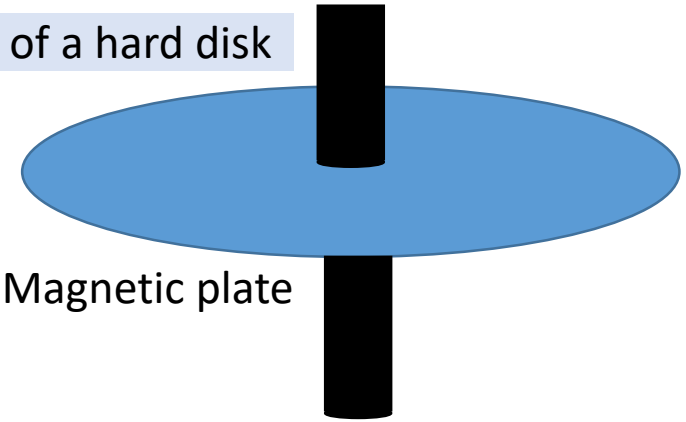
Developing with Visual Studio Code and creating an virtual environment

File System: folders and files



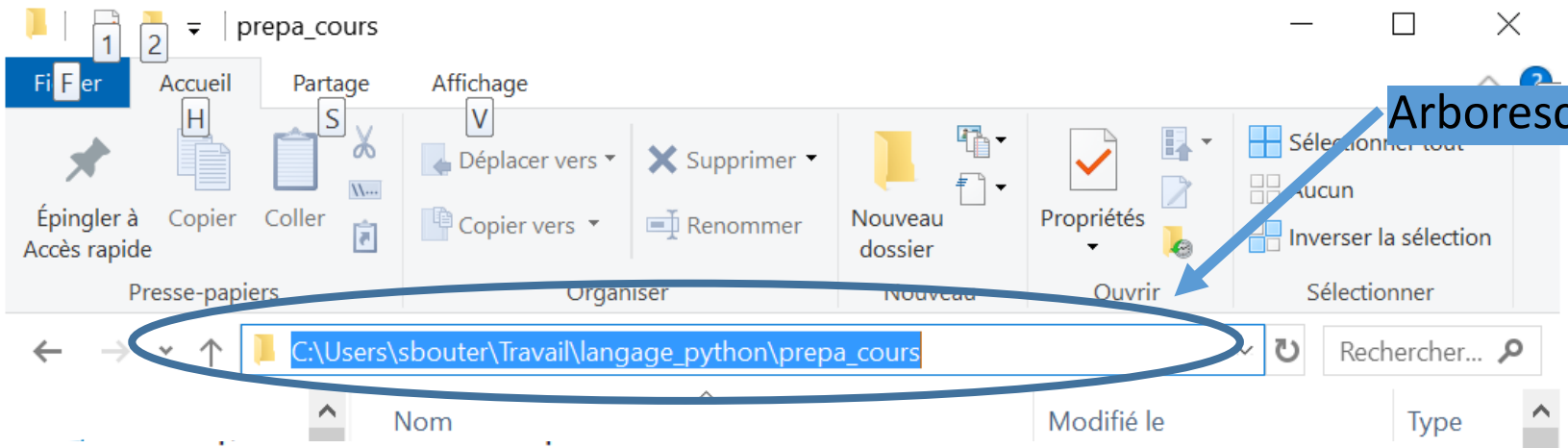
Non-volatile storage for data and programmes

Structure of a hard disk



Flash memory based on semi-conductors : USB sticks, SD card…

Folders: how to handle them



Arborescence of folders

```
import os

print(os.getcwd())
os.chdir('C:\\Users\\sbouter\\Travail\\langage_python\\prepa_cours')
print(os.getcwd())

ShaskespeareQuote = "There's not a note of mine that's worth the noting"

outfile = open('destinationFile.txt','w')
outfile.write(ShaskespeareQuote)
outfile.close()
```

Print the current folder

Change the folder

Name of the file

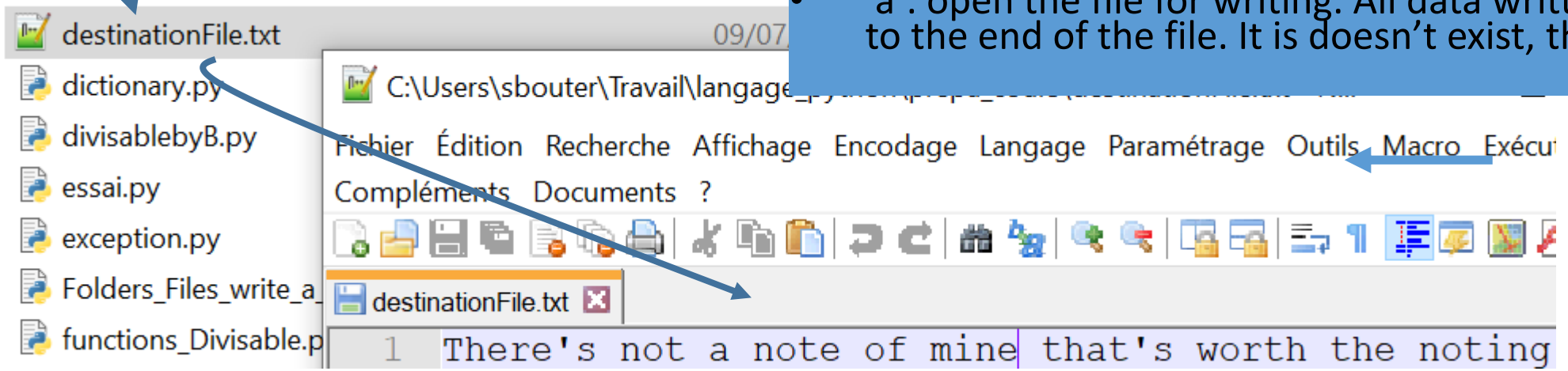
How to get access to files

```
import os

print(os.getcwd())
os.chdir('C:\\Users\\sbouter\\Travail\\langage_python\\prepa_cours')
print(os.getcwd())

ShaskespeareQuote = "There's not a note of mine that's worth the noting"

outfile = open('destinationFile.txt', 'w')
outfile.write(ShaskespeareQuote)
outfile.close()
```

This programme is meant to copy a string of characters into a file. In this the file is created in the folder as the Python programme

Mode of access to the file:

- 'w': open the file for writing. If the file exists, the content is erased. It is doesn't exist, the file is created.
- 'r': open the file for reading only.
- 'a': open the file for writing. All data written will be appended to the end of the file. It is doesn't exist, the file is created.



Copying the content of a file in another one

Mode of access to the files:
The file “sourceFile.txt” is opened for reading and “destinationFile.txt” for writing. The variables “inFile” and “outFile” reference the file objects.

```
inFile = open('sourceFile.txt','r')
outFile = open('destinationFile.txt','w')
temp = inFile.read()
outFile.write(temp)
inFile.close()
outFile.close()
```

Close the files

Read the content of the file “sourceFile.txt” and store in the variable “temp”

Write the content of the variable “temp” into the file “sourceFile.txt”

Numpy library

"NumPy" is a specialised library for the Python programming language intended to deal with multi-dimensional arrays and matrices, along with a large set of functions to operate on these arrays.
This library is often used in many other library openCv (image processing), matplotlib (charts, curves…), pypot(intended for the controlling the robot)….
The example below shows few possibilities of "numpy" such the cropping.

```
import numpy as np

a = np.array(range(49)).reshape(7,7)
print(a)
b = a[1:5,2:4]
print(b)
print(b.size)
print(np.sum(b))
```

```
[[ 0  1  2  3  4  5  6]
 [ 7  8  9 10 11 12 13]
 [14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34]
 [35 36 37 38 39 40 41]
 [42 43 44 45 46 47 48]]
```

2:4

```
[[ 9 10]
 [16 17]
 [23 24]
 [30 31]]
```

1:5

```
8
160
```

For column : numpy_Array_name[: ***,column***]
For row : numpy_Array_name[***row,*** :]

Numpy, dealing with the matrixes

$$(S_3)\begin{cases} 2x+4y-2z=4 \\ x+3y+z=10 \\ 3x-y+z=4 \end{cases}$$

Solutions of a System of linear equations

```
import numpy as np

a = np.array([[2, 4, -2], [1, 3, 1], [3, -1, 1]])
print(a)
b= np.array([4, 10, 4])
b.shape = (3,1)
print(b)
print(np.linalg.inv(a))
x = np.matmul(np.linalg.inv(a),b)
print(x)
```

```
[[ 2  4 -2]
 [ 1  3  1]
 [ 3 -1  1]]
[[ 4]
 [10]
 [ 4]]
[[ 0.11111111 -0.05555556  0.27777778]
 [ 0.05555556  0.22222222 -0.11111111]
 [-0.27777778  0.38888889  0.05555556]]
[[1.]
 [2.]
 [3.]]
```

Numpy arrays and lists

```
import numpy as np

myList1=
[[10,2],[11,3],[12,4],[14,5],[15,6],[16,7],[17,8]]
myList2= [[10,2,30],[11,3,31],[12,4,32]]
myArray1 = np.array(myList1)
print(myArray1)
print(myArray1[:,0])
print(myArray1[:,1])
print(myArray1[0,:])
print(myArray1[1,:])
myArray2 = np.array(myList2)
print(myArray2)
print(myArray2[:,0])
print(myArray2[:,1])
print(myArray2[:,2])
print(myArray2[1,:])
```

```
[[10  2]
 [11  3]
 [12  4]
 [14  5]
 [15  6]
 [16  7]
 [17  8]]
[10 11 12 14 15 16 17]
[2 3 4 5 6 7 8]
[10  2]
[11  3]
[[10  2 30]
 [11  3 31]
 [12  4 32]]
[10 11 12]
[2 3 4]
[30 31 32]
[11  3 31]
```

Matplotlib: plot

Matplotlib is a library a function intended to give tools to draws charts, curves and histograms

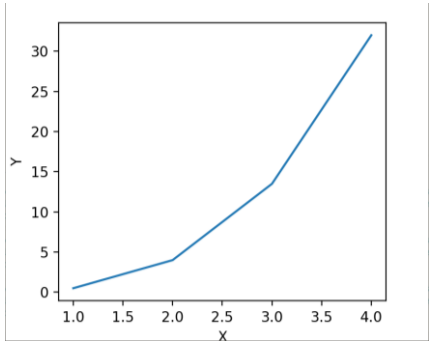
Installing the libraries needed

python -m pip install pip or python –m install - -upgrade pip

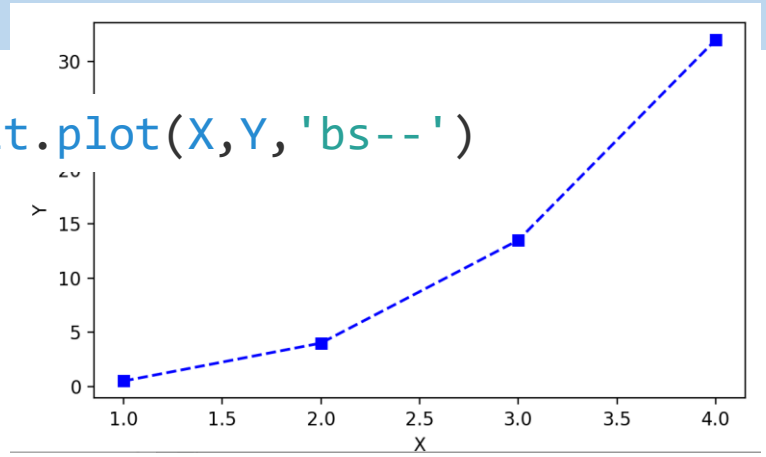
python -m pip install matplotlib

The command “python”can be either “py” or “python3X”, it depends on the symbolic link

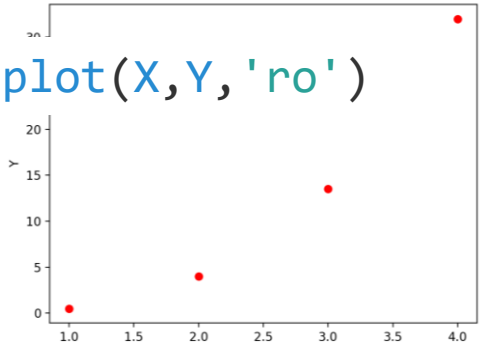
```
import matplotlib.pyplot as plt
Y =  [0.5, 4, 13.5, 32]
X= [1, 2, 3, 4]
plt.plot(X,Y)
plt.ylabel('Y')
plt.xlabel('X')
plt.show()

#The default curve is a solid blue line
```



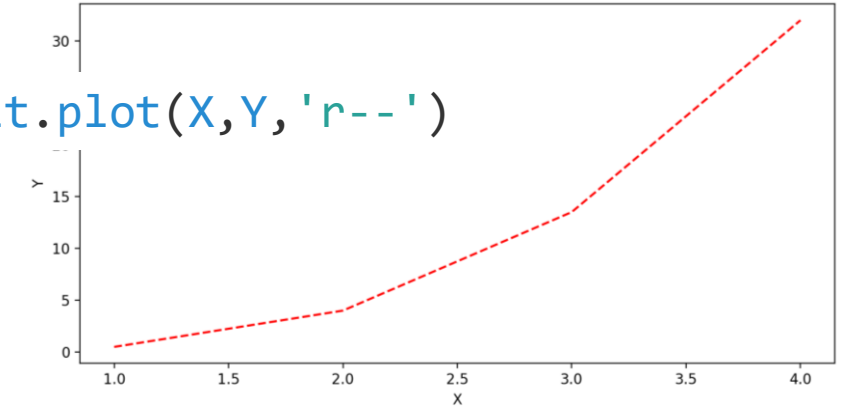

```
plt.plot(X,Y,'bs--')
```



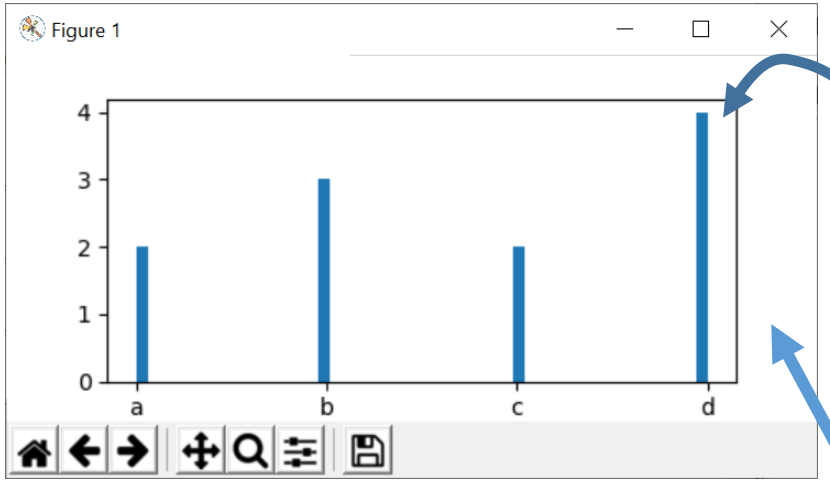

```
plt.plot(X,Y,'ro')
```

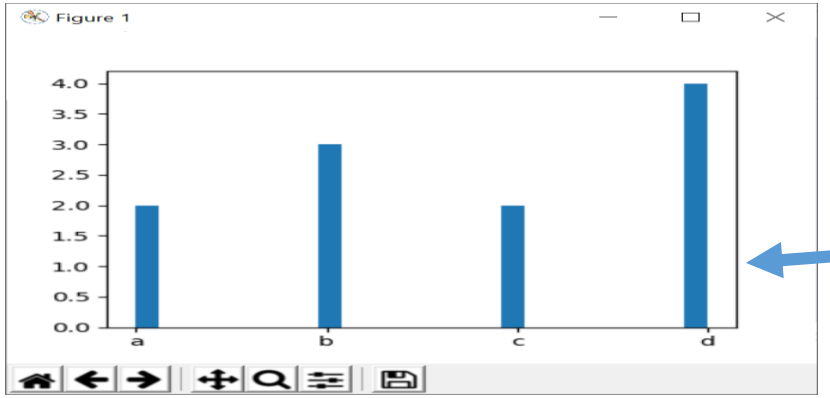



```
plt.plot(X,Y,'r--')
```



matplotlib: Histograms





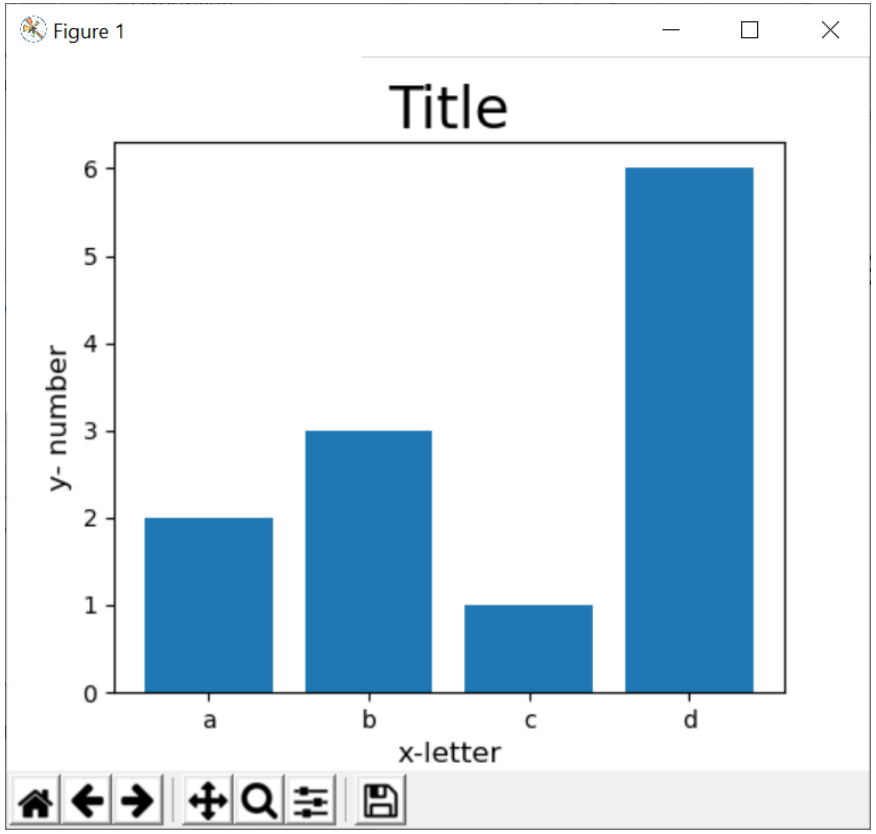

```
import matplotlib.pyplot as plt
# Plot Histogram on x
# elements to count and sort made up of four differentsletters
x = ['a','a','b','b','b','c','c','d','d','d','d']
# 2 ‘a’, 3 ‘b’, 2 ‘c’ and 4 ‘d’
```

```
plt.hist(x, bins=50) # building the histogram
plt.show()
```

“bins” is an integer and it defines the number of equal-width bins in the range.

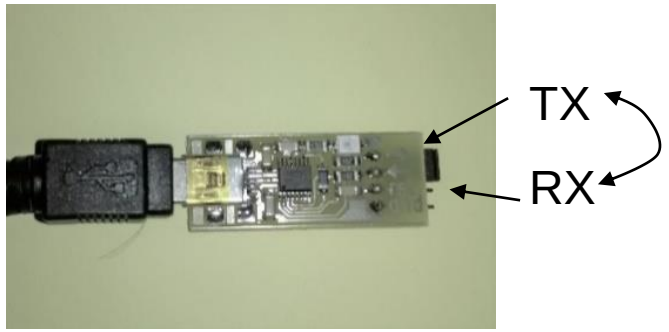
```
plt.hist(x, bins=25)
plt.show()
```

Matplotlib: Bar charts

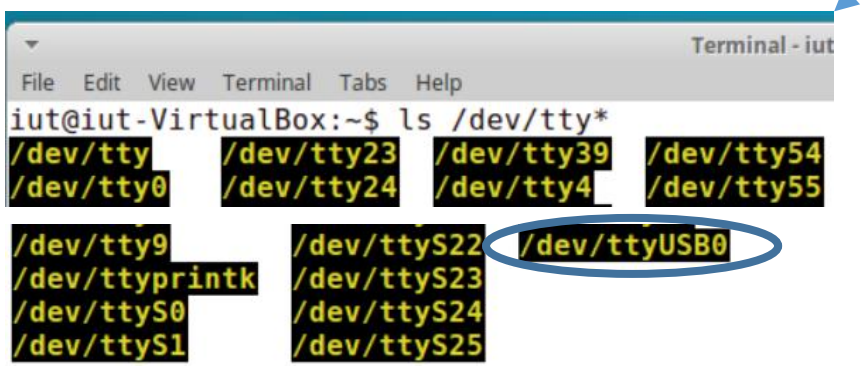



```
import matplotlib.pyplot as plt

x = ['a','b','c','d']
y = [2,3,1,6]
plt.close('all')
fig, ax = plt.subplots()

ax.bar(x,y )

ax.set_xlabel('x-letter', fontsize= 12)

ax.set_ylabel('y- number', fontsize= 12)

ax.set_title('Title', fontsize= 24)

plt.show()
```

Asynchronous serial link: example of connection for a first exercise



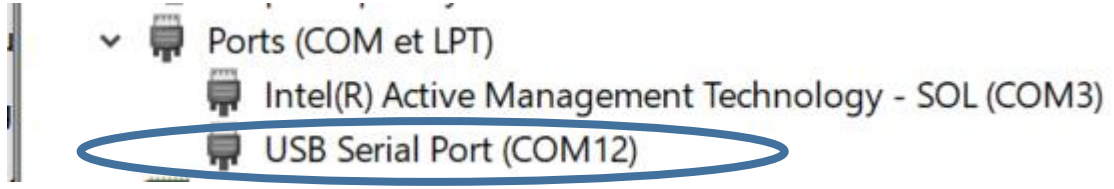
The pins RX and TX are connected to each other



With Linux, when you have plugged thus board to the USB connector of the PI board, check the port which appears by running the following command: $ ls /dev/tty*.

A virtual port is defined over USB

With Windows, check the port which appears by looking into the peripheral manager



serialpy a library dealing with the serial port

```
import time #the needed libraries
import serial

sendCommand = bytearray(6)
receivedData = bytearray(6)

sendCommand[0] = ord('R')
sendCommand[1] = 65
sendCommand[2] = 66
sendCommand[3] = 67
sendCommand[4] = 68
sendCommand[5] = ord('\n')

# the programme is opening a virtual port defined over USB
h1 = serial.Serial("COM12",baudrate = 9600,timeout = None) # no timeout

test = 0
while test != 'F':
    h1.write(sendCommand) #each character is stored into a byte
    receivedData = h1.readline()
    print("data received",receivedData)
    print("enter F to finish")
    test = input()
h1.close()
```

Opening the selected port. This function return an abstract reference to a resource, here the port ttyUSB0, called an handle

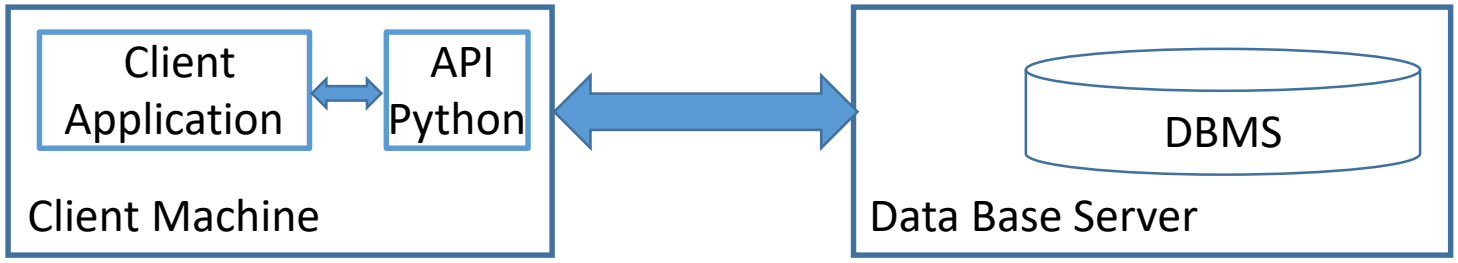
Sending a message

Waiting for data

Closing the port

```
= RESTART: C:/Users/sbouter
LinkSimpleExamplePySerialLi
data received b'RABCD\n'
enter F to finish
A
data received b'RABCD\n'
enter F to finish
Z
data received b'RABCD\n'
enter F to finish
E
data received b'RABCD\n'
enter F to finish
Y
data received b'RABCD\n'
enter F to finish
F
>>>
```

MySql Database



Installing mysql connector:
C:\python38-64>python -m pip install mysql-connector-python

how to access to the content a Database

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="m2gsat"
)


mycursor = mydb.cursor()

mycursor.execute("SHOW TABLES")
for x in mycursor:
  print(x)

mycursor.execute("SELECT * FROM exam1")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)

mycursor.close()
```

Table
exam1
student

```
('exam1',)
('student',)
```

←T→	id	title	date	id_student
Modifier Copier Effacer	1	math	2021-04-12	*NULL*
Modifier Copier Effacer	2	electronique	2021-05-01	*NULL*
Modifier Copier Effacer	3	meca	2021-06-01	*NULL*

```
(1, 'math', datetime.date(2021, 4, 12), None)
(2, 'electronique', datetime.date(2021, 5, 1), None)
(3, 'meca', datetime.date(2021, 6, 1), None)
```

Inserting a record

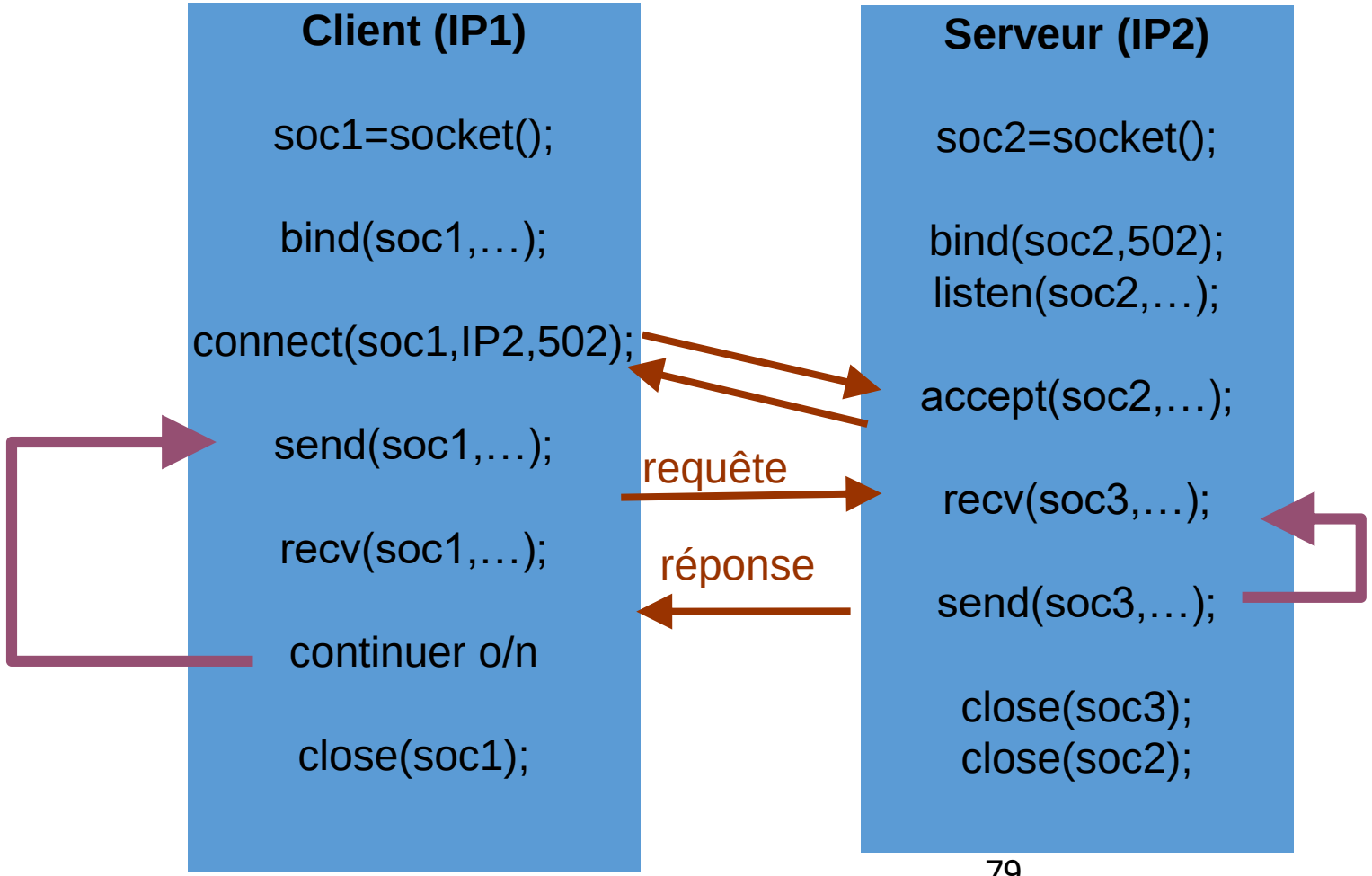
```
mycursor.execute("SELECT * FROM exam1")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)

sql = "INSERT INTO exam1 (id, title, date, id_student) VALUES (NULL,%s, %s, NULL)"
val = ("physique","2014-09-10")
mycursor.execute(sql, val)
mydb.commit()


mycursor.execute("SELECT * FROM exam1")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)


mycursor.close()
```

```
(1, 'math', datetime.date(2021, 4, 12), None)
(2, 'electronique', datetime.date(2021, 5, 1), None)
(3, 'meca', datetime.date(2021, 6, 1), None)
(1, 'math', datetime.date(2021, 4, 12), None)
(2, 'electronique', datetime.date(2021, 5, 1), None)
(3, 'meca', datetime.date(2021, 6, 1), None)
(5, 'physique', datetime.date(2014, 9, 10), None)
```

Communication Interface : the sockets

- The sockets interface provides primitives (functions) necessary for communications over TCP/IP networks
- Sockets are access point to the network by which a program can send or receive data
- Client and server applications only see the communication layers through the API (Application Programming Interface) socket.
- Access to the network is provided at the Network and transport layers.
- The sockets interface, initially associated with the UNIX operating system, has spread to other OS, including Windows

connection, data transmission and closing the connection



When running on the same host, the loop address 127.0.0.1 can be used. This address is very useful to test network applications even if the computer is not connected to any network

The Server

```
import socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # managing the
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
server.bind(('', 17000)) # the application is responsive on the port number 17001
buff_size = 100
server.listen(10)
while True:
        while True:
                response = b''
                print("waiting for connection")
                client, address = server.accept() # waiting for a connection
                # create another socket intended for the data transmission
                print ("{} connected".format( address ))
                response = client.recv(buff_size)
                print("message received=", response)
                client.sendall(b'I am the serveur')
                print ("Closing")
                client.close() # closing the socket intended to recieve/send data from/to the
remote host
server.close() # closing the scoket intended to mange connection
```

The Client

```
import socket
response = b''
buff_size = 100
connexion_serveur = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
try:
    connexion_serveur.connect(("127.0.0.1", 17000))
    connexion_serveur.sendall(b'temperature')
    response = connexion_serveur.recv(buff_size)
    print(response)
finally:
    connexion_serveur.close()
```

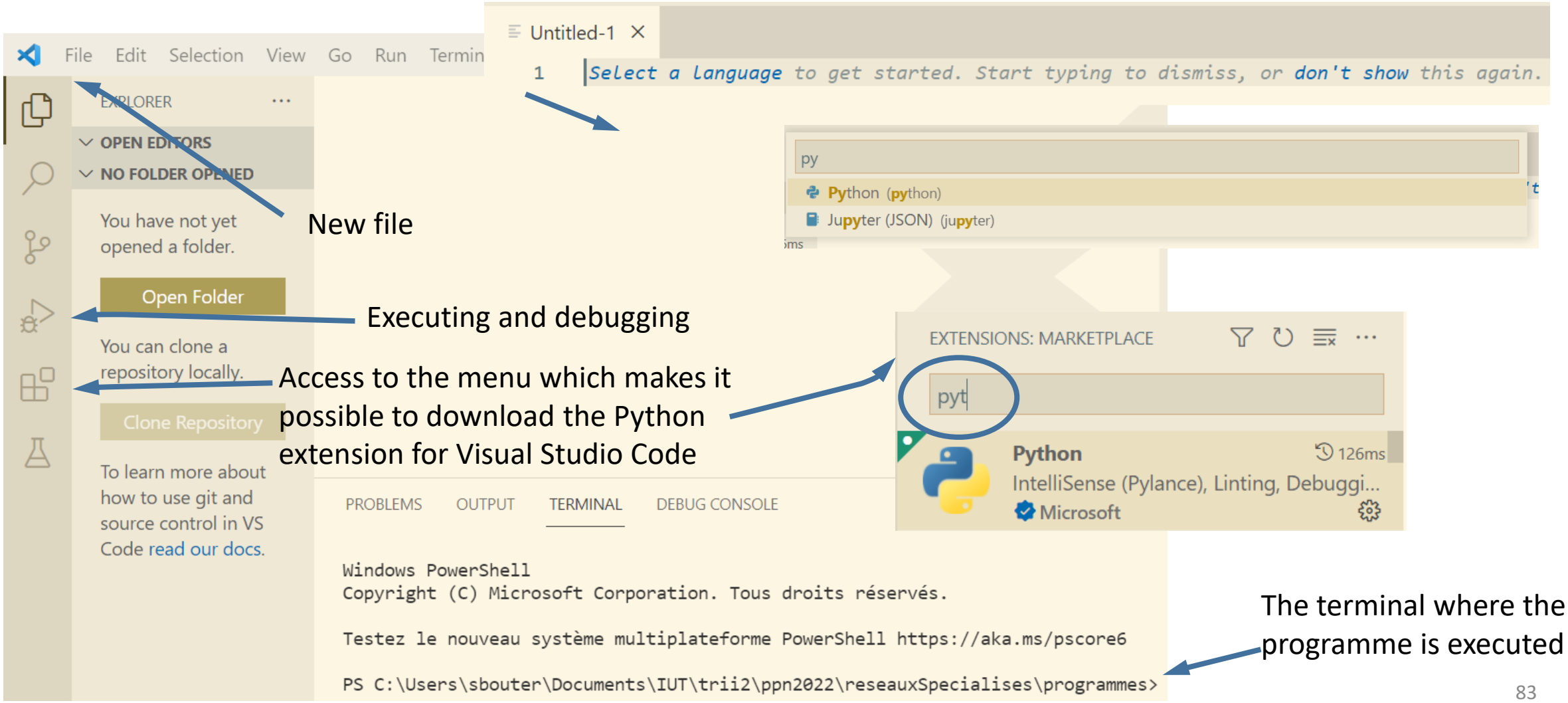
Converting : string <----> array of bytes

The data exchanged is of type bytes. In Python, byte strings (bytes) are represented as character strings (limited to ascii encoding), preceding with a b : b‘hello!'

•Conversion str→ bytes : bytes("donnée", "utf-8") → b'donn\xc3\xa9e'

•Conversion bytes → str : b'donn\xc3\xa9e'.decode() → "donnée"

Development Environment: Visual Studio Code



New file

Executing and debugging

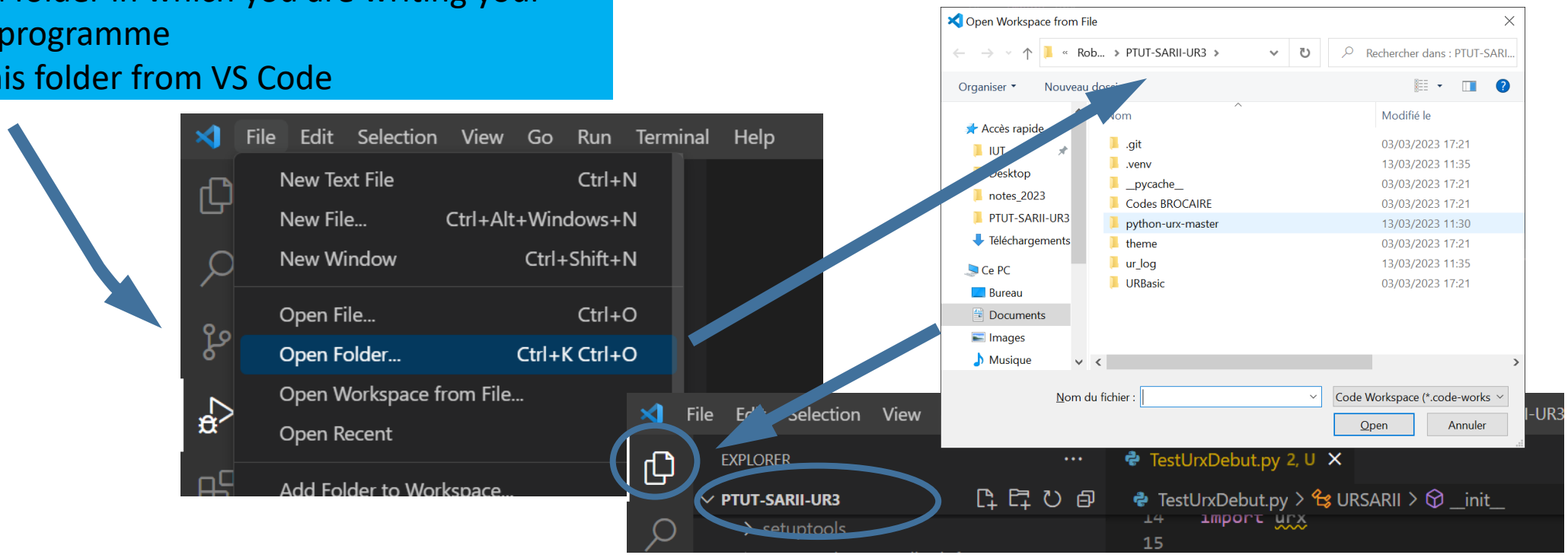
Access to the menu which makes it possible to download the Python extension for Visual Studio Code

The terminal where the programme is executed

Creating a virtual environment in VSCode(1)

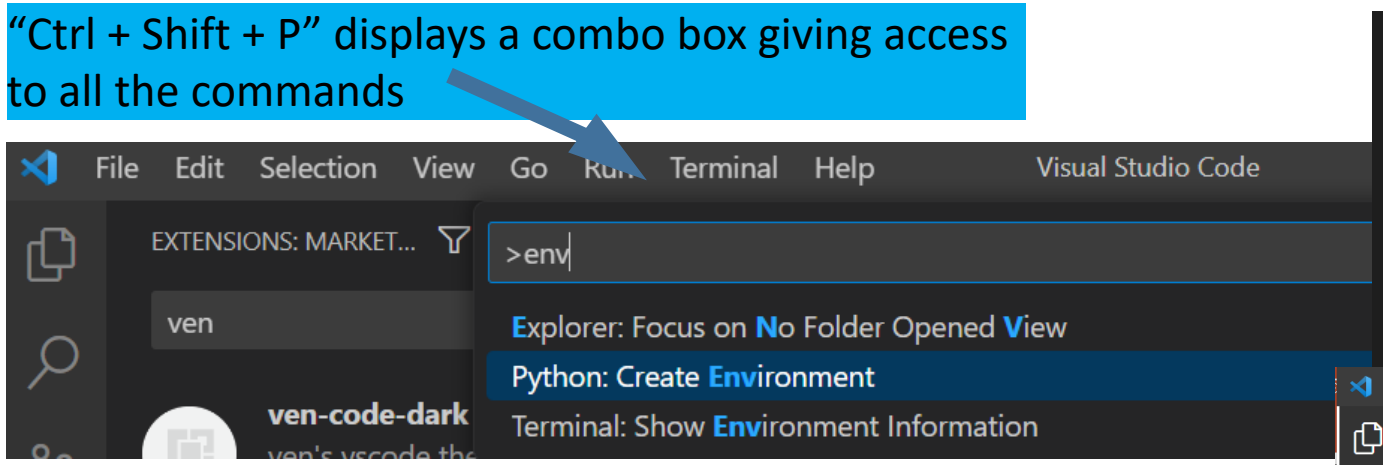
- A virtual environment create an environment which allows you isolate the packages you install per workspace.
- A virtual environment creates a folder that contains a copy (or symlink) to a specific interpreter.
- When you install a package into a virtual environment it will end up in this new folder so that there are not link with other packages used by other workspaces.

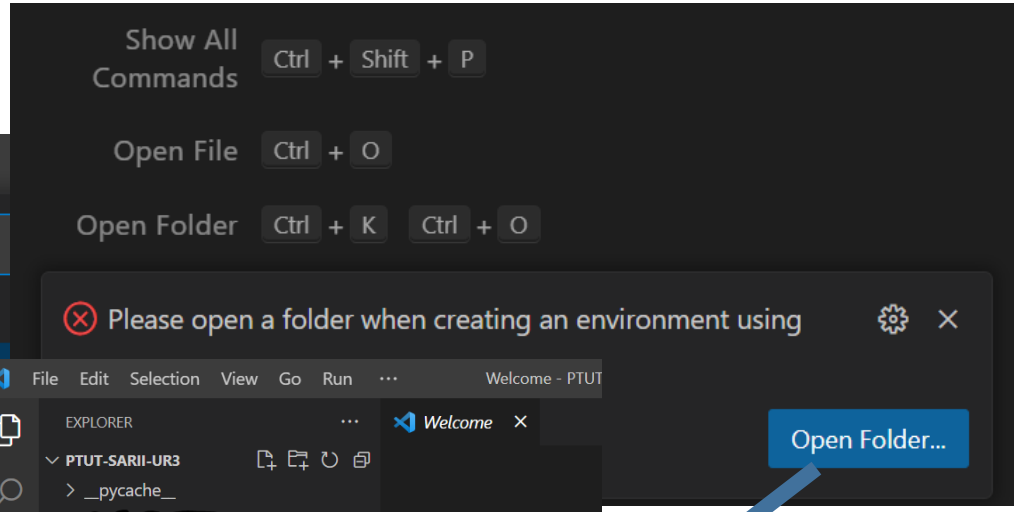
- Create a folder in which you are writing your Python programme
- Open this folder from VS Code

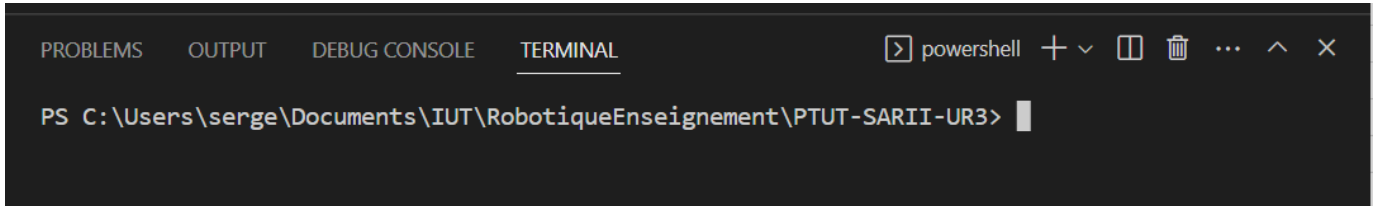


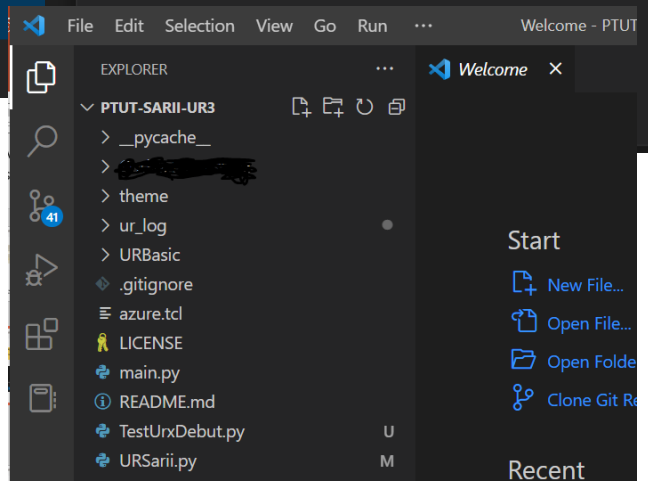
Creating a virtual environment in VSCode(2)

"Ctrl + Shift + P" displays a combo box giving access to all the commands



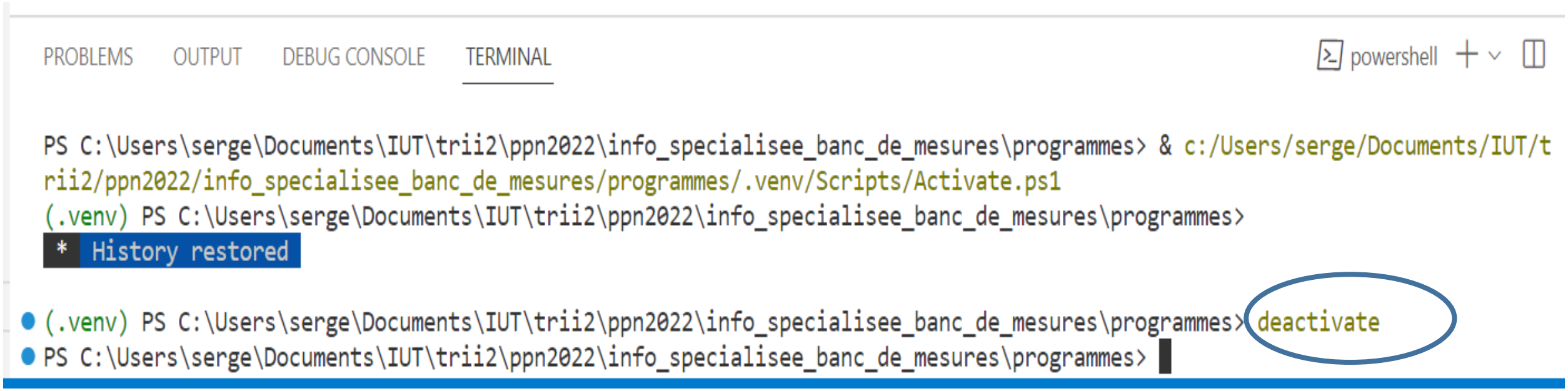






Getting the list of the libraries installed in the virtual environment:
…>python -m pip list
Or
$ python -m pip list

Disable the virtual environment



Assignement exercises

The exercises are dealt with to support the lectures. They allows the students to apply the concept seen during the lectures, get familiar with the algorithm and be trained on how the convert a problem into algorithm. Exercises proposed:

The cash dispenser

Checking whether a string of characters matches a number (string, tuple, for and while statement and function)

Nought and cross

If statement and operators

- The cash dispenser

A cash dispenser only provides 50 and 10 euro notes. The customer enters an amount in euros.

1) You are asked to write a programme which informs the customer of the possibility to withdraw the amount in 50 euro notes only then in 10 euro notes only. The amount asked is an integer.
2) You have to modify the previous programme so that it gives the number of 50 euro and 10 euro notes, when possible, the costumer can withdraw. A priority is given to 50 euro notes, when possible

If statement and string of characters

Checking whether a string of characters matches a number:

The programme must:

- makes it possible entering a string of characters,
- Add to a variable, initialised to 0, the number of occurrences of the characters “0”, “1”, “2”, “3”,…”8”, “9” and “.” in the string characters.
- Compare the content of the variable with the length of the string of characters
- Displays whether the string matches a number. You must add a part which specifies whether the string matches a float or integer.
- Comments:
  - you can get the length of a string with the function “len”.
  - The number of a character in a string can be obtain with the function “count”

Accessing a character in a string of characters or an element in a tuple.

Checking whether a string of characters matches a number:

1) You must modify the previous programme so that the latter performs from the initialised string “0123…89.” it increments a variable, initialised to 0, at the occurrence of one of the elements of the string “0123…89.” in the formerly entered string of characters.

2) You must modify the previous programme so that the latter performs from a initialised tuple containing the elements “0”, “1”, “2”, “3”,…”8”, “9” and “.”.

Accessing a character in a string or an element in a tuple and the “for” and while statement

Checking whether a string of characters matches a number:

1) You are asked to modify the previous programme so that the latter performs from the initialised string “0123…89.” it increments a variable, initialised to 0, at the occurrence of one of the elements of the string “0123…89.” in the formerly entered string of characters. Instead of writing successive “if” statements, you must implement a control flow “for”

2) You are asked to modify the previous programme so that the latter performs from a initialised tuple containing the elements “0”, “1”, “2”, “3”,…”8”, “9” and “.”.

3) You are asked to modify the previous programme by replacing the “for” statement with a “while” statement

Functions

Checking whether a string of characters matches a number:

You are asked to implement the test inside a function. The function returns the result of the test as below. You also write a programme which the function as below.

Calling the function

```
def numberTest(NbToTest):
    ...
    ...
    ...
    ...

a= input("enter a number = ")

b = numberTest(a)

print ("type of number =" ,b)
```

Running the programme

```
enter a number = 567
type of number = <class 'int'>
>>>
= RESTART: C:\Users\sbouter\Travail\
umber.py
enter a number = 4.78
type of number = <class 'float'>
>>>
= RESTART: C:\Users\sbouter\Travail\
umber.py
enter a number = 5N7
type of number = False
>>>
```

Nought and cross

- You are asked to write a programme which manages the game “nought and cross”. A basis is given below and it mostly consists of a function “buildArray” intended for drawing the grid and filling it with noughts and crosses.

```
def buildArray(combination):
    ''' this function is intended for drawing the grid and fill it with noughts and crosses
        according to the content of a matrix 3X3. the elements must assigned to either 'O' or 'x' or ' '
        '''
    print("  1  2  3")
    print("  _________")
    print('1| '+str(combination[0][0])+'| '+ str(combination[0][1])+'| ' + str(combination[0][2])+ '|')
    print(" |__|__|__|")
    print('2| '+str(combination[1][0])+'| '+ str(combination[1][1])+'| ' + str(combination[1][2])+ '|')
    print(" |__|__|__|")
    print('3| '+str(combination[2][0])+'| '+ str(combination[2][1])+'| ' + str(combination[2][2])+ '|')
    print(" |__|__|__|")
''' end of the function '''

noughtsAndCross = [[' ',' ',' '],[' ',' ',' '],[' ',' ',' ']]

buildArray(noughtsAndCross)
```

```
= RESTART: C:\Users\
htdAndCross.py
  1  2  3
  _________
1|  |  |  |
 |__|__|__|
2|  |  |  |
 |__|__|__|
3|  |  |  |
 |__|__|__|
premier joueur
row = 1
column = 3
  1  2  3
  _________
1|  |  | O|
 |__|__|__|
2|  |  |  |
 |__|__|__|
3|  |  |  |
 |__|__|__|
deuxieme joueur
row = 3
column = 3
  1  2  3
  _________
1|  |  | O|
 |__|__|__|
2|  |  |  |
 |__|__|__|
3|  |  | x|
 |__|__|__|
premier joueur
row =
```

Practical Work Exercices on the language

The if statement , the tuples and the while statement

- Write a programme which allows the operator to enter the year and tests if the year is a leap year. You are asked to find the algorithm to get the leap year.
- you are asked to take up the previous programme to write a function “IsALeapYear” which tests the year and returns a boolean; the year (an integer) is the input argument of the function. Write a programme to test the function.
- Write a programme which allows the operator to enter the day, the month and the year and calculates the date of the day after. We suppose the input of the date is correct and does not need to be checked. The number of days in a month can be stored in a tuples (31,28,31,…30,31)
- Write a programme calculating the highest common factor. You are advised to use the Euclid algorithm.

The while statement and the list

- Write a programme*(UK)* that allows to input series of positive numerical values. This programme determines and displays the maximum at each input. The programme *(UK)* finishes when a negative value is input.
- Modify the previous programme to perform the sum of all the input numbers except the last one, the negative number.
- Modify the previous programme so that the input numbers, except the last one are stored in a list object. The programme must also perform the sum of the numbers of the list using the method “sum” associated with “list”
- Write a programme that for an amount given in euros (without cents) displays the minimal number of banknotes and coins needed to reach the amount. Example: 2849 = 5 ∗500 + 1 ∗200 + 1 ∗100 + 2 ∗20 + 1 ∗5 + 2 ∗2

  notices:

  the quotient and the remainder of the integer division are obtained respectively by the statements “a//b” and “a mod b”, with a and b as integer.

  The values of the notes and coins of the currency can be stored in a tuples.

For statements and algorithms on the integers

In this exercise we will discover an interesting property of the number 6174.

Let N1 be a positive integer of 4 digits. Form from its digits two numbers: the larger one with its digits written in decreasing order and the smaller one with its digits written in increasing order. Calculate the difference N2 between these two numbers. Treat N2 in the same way and form the sequence N1,N2,N3, ...Ni. We then have the following two properties:

1. The number 6174 is an invariant, i.e. 7641 - 1467 = 6174;
2. 2. All other four-digit numbers lead to the same result after n operations with n = …..

Example :N1 = 3015

5310 - 0135 = 5175 = N2

7551 - 1557 = 5994 = N3

9954 - 4599 = 5355 = N4

5553 - 3555 = 1998 = N5

9981 - 1899 = 8082 = N6

8820 - 0288 = 8532 = N7

8532 - 2358 = 6174 = N8

Note: Numbers consisting of 4 identical digits should be excluded.

Write a program in that asks the user for an integer of 4digits and displays the sequence N1,N2,N3, ... ,Nn.

The for statement and the functions

- Write a programme calculating the factorial of an integer. You must use the “for” statement.
- you are asked to take up the previous programme to write a function which calculates and returns the factorial of a integer; the integer is the input argument of the function. Write a programme to test the function.
- you are asked to write a function which calculates and returns the Highest Common Factor of two integers. These integers are the input arguments of the function. Write a programme to test the function.
- Write a programme allowing a fraction to be simplified. You are advised to use the function calculating the Highest Common Factor.

Dealing with the string of characters

- Write an programme that displays the letters of the alphabet. You must use a “for” statement to display each letter of the alphabet. You are also asked to look into the functions chr() and ord().
- Modify the programme to store the letter of the alphabet in a list and display the content of the list

```
==== RESTART: C:/Users/sbouter/Travail/langage_python/exercices/alphabet.py ====
alphabet =  ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

- Write an programme :
  - makes the inputting of string of characters
  - makes the inputting of one character. If several characters have been entered, only the first character is taken into account
  - displays the number of occurrences of this character in the string of characters. You are asked to look into the method “count” associated with the object string.

```
Enter a message :fjghdjhgdjkhgdjk
Enter a letter : hfghg

nb of  letter h  =  3
```

Exercise: statistical calculations on a text(1)

We want to perform statistical calculations on the content of a string of characters. For this you are asked to write a programme which:

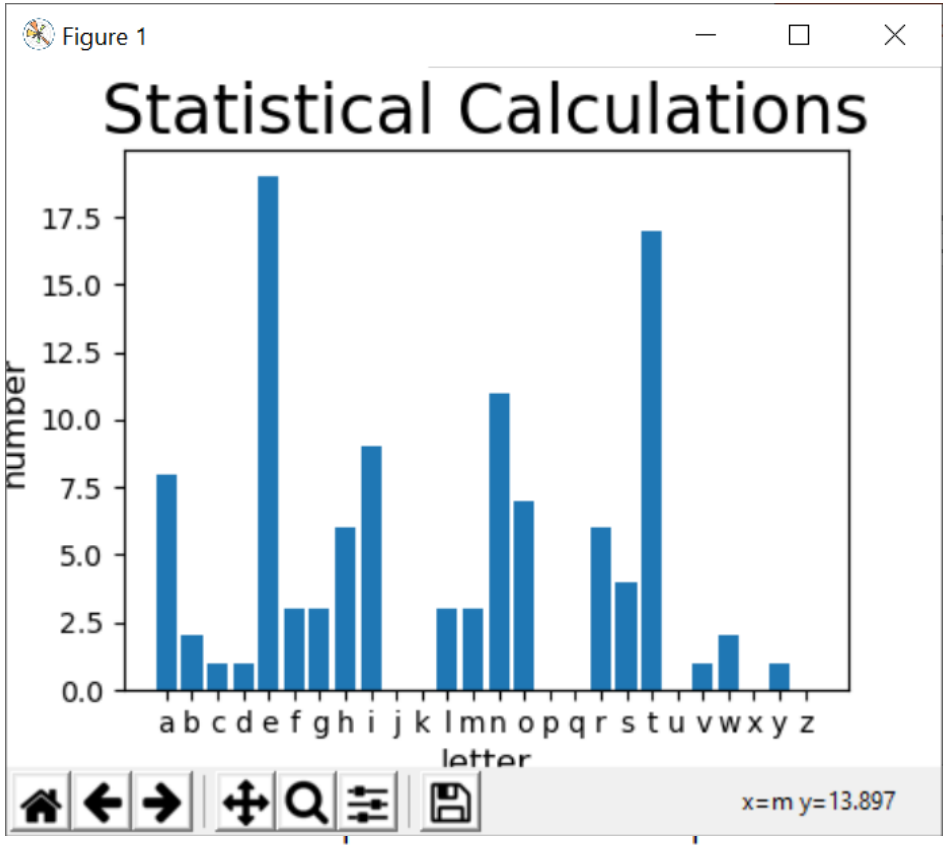
- determine the number of words,
- counts the occurrence of a given character,
- determines, displays the number of occurrences for all letters (of the alphabet) and calculate the number of occurrences for the character with the most occurrences in the text,
- draws a "graph" visualizing the occurrences of all the letters of the alphabet in the text file. The graph will be represented as on the right.

In this case the string of characters used is below.

```
#There's not a note of mine that's worth the noting
ShaskespeareQuote = "There's not a note of mine that's worth the noting"
```

```
a  = **
b  =
c  =
d  =
e  = *****
f  = *
g  = *
h  = ****
i  = **
j  =
k  =
l  =
m  = *
n  = *****
o  = *****
p  =
q  =
r  = **
s  = **
t  = *******
u  =
v  =
w  = *
x  =
y  =
z  =
```

Exercise: statistical calculations on a text(2)



It is the same exercise as the previous one but the displaying of the result (see on the left) is based on the library “matplotlib”. You can find an example in the course. You are asked to display the result on a chart as on the left.

In this case the string of characters used is below.

Now you are asked to perform the calculation from the content of a file. For that, the programme must read a file and assigns it to a variable.

```
#There's not a note of mine that's worth the noting
ShaskespeareQuote = "There's not a note of mine that's worth the noting"
```

Object oriented programming

Class

Instantiation

Inheritance

Overloading the operator

Class Variable

Graphical User Interface Tkinter

The objects in Python language

A class is a complex type defined by the programmer. A class is made up of data and methods. The purpose of the method consists in processing data of the class.
This way implements a tight association between data and methods. Usually, only the methods can have access to the data of the class. The definition of a class can be picked out by the key-word class and consists in a list of attributes and of methods.
In the Object Oriented Paradigm, the declaration of variables of a given class is called instantiation and the variables are then qualified as object

The Language has got native class such as string, Boolean, list…. These classes are a set of attributes and methods. The attributes characterizes an object instantiated from a class. For instance, the main attribute of the class string is a array of characters. The methods operates on the attributes, to set the value of an attribute, get its value, the length of the string of characters, concatenate two strings of characters….
The Python Language makes it possible to create new types of class.

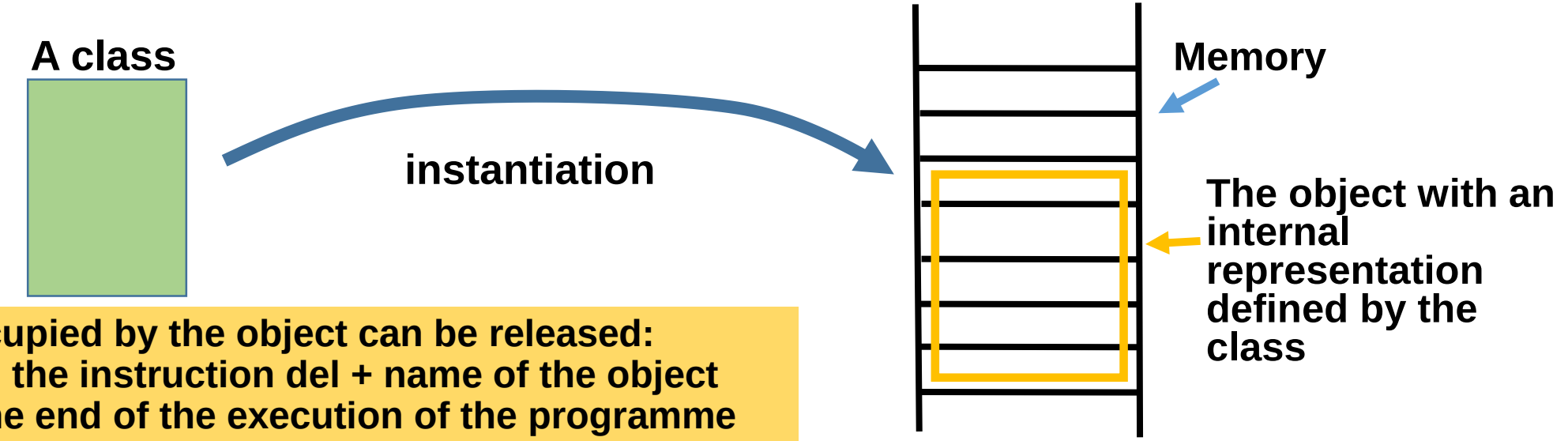
Class, instanciation and object

class

Attributes defining the object and its state

Methods accessing the attributes of the class

treatment: carrying out the treatments associated with the object. The treatments can get the content of the attributes and can modify the value of these attributes
getter: getting the content of an attribute
setter: setting a value to an attribute



The memory occupied by the object can be released:
- **Explicitly with the instruction del + name of the object**
- **Implicitly at the end of the execution of the programme thanks to the "Garbage Collector"**

Defining and instantiating a class: example

```
class MyVector(object):
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z
    def Norm(self):
        return ( self.x * self.x + self.y * self.y + self.z * self.z )**0.5

v1 = MyVector(1,2,3)
print(v1.Norm())
```

A class "MyVector"

Attributes of the class.

The input parameter is the object in question

Initialisation of the attributes

Method "Norm"

The object "v1" instantiated from the class "MyVector"

Calling the method "Norm" associated with the object. The calling is done by mentioning the name of the object + a dot + name of the method

```
== RESTART: C:\Users\sbouter\
3.7416573867739413
>>>
```

The key-word "__init__" is referenced in the Python language as the name of a method and is commonly known as "constructor". The constructor defines the way the object must be instantiated, in particular the initialization of the attributes

Dealing with the attributes inside the class: self

Attributes of the class are suffixed by the key word "self"

A class "MyVector"

The input parameters are intended to initialize the attributes of the object

```
class MyVector(object):
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z
    def Norm(self):
        return ( self.x * self.x + self.y * self.y + self.z * self.z )**0.5

v1 = MyVector(1,2,3)
print(v1.Norm())
```

Initialisation of the attributes

The input parameter is the object in question

Method "Norm"

The access to the attributes can be done using "self" + dot + name of the attributes

The key word "self" represents any instance of a class

In this case,the key word "self" refers to the "MyVector" class object "v1"

Other explanations and adding another method

The class "MyVector" inherits the basic attributes and methods of the language contained in "Object"

```
class MyVector(object):
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z
    def Norm(self):
        return ( self.x * self.x + self.y * self.y + self.z * self.z )**0.5

    def ScalarProduct(self,other):
        return ( self.x * other.x + self.y * other.y + self.z * other.z )

v1 = MyVector(1,2,3)
v2 = MyVector(-2,1,3)
print(v1.ScalarProduct(v2))
print(MyVector.ScalarProduct(v1,v2))
```

The "Scalar Product" needs 2 input parameters

Method "ScalarProduct"

Calling the method "ScalarProduct" associated with the object "v1". The method needs a second input parameter

Calling the method "ScalarProduct" associated with the class "MyVector". The method needs two input parameters

Accessing to the attributes and default arguments

It can be useful to define values by default for the attributes

```
class MyVector(object):
    def __init__(self,x=0,y=0,z=0):
        self.x = x
        self.y = y
        self.z = z
    def Norm(self):
        return ( self.x * self.x + self.y * self.y + self.z * self.z )**0.5

v1 = MyVector()
print(v1.Norm())

print(v1.x)
print(v1.y)
print(v1.z)
```

If no input parameters are given, Python takes the values defined by default. If there were non default arguments, then Python would give an error

```
== RESTART: C:\Users\
0.0
0
0
0
>>>
```

The attribute x can be accessed outside the class. The dot notation is used to access to the attribute.

```
v1 = MyVector(1,2,3)
print(v1.Norm())
v1.x  = 34
print(v1.x)
```

```
========================= RESTART:
3.7416573867739413
34
```

Displaying the content of an object

```
class MyVector(object):
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z
    def Norm(self):
        return ( self.x * self.x + self.y * self.y + self.z * self.z )**0.5

    def ScalarProduct(self,other):
        return ( self.x * other.x + self.y * other.y + self.z * other.z )

    def __str__(self):
        return "(" + str(self.x) + "," + str(self.y) + "," + str(self.z) + ")"

    def VectorProduct(self,other):
        temp = MyVector(0,0,0)
        temp.x =  self.y * other.z - self.z * other.y
        temp.y =  self.z * other.x - self.x * other.z
        temp.z =  self.x * other.y - self.y * other.x
        return temp

v1 = MyVector(1,2,3)
v2 = MyVector(2,4,6)
print(v1.VectorProduct(v2))
print(MyVector.VectorProduct(v1,v2))
```

The name "__str__" is referenced in the Python language as the name of a method. This method must be defined by the programmer. It makes it possible to display the content of the object the way chosen by the programmer

This method returns a result which is "MyVector" type object.

Overloading an operator

```
class MyVector(object):
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z
    def Norm(self):
        return ( self.x * self.x + self.y * self.y + self.z * self.z )**0.5

    def ScalarProduct(self,other):
        return ( self.x * other.x + self.y * other.y + self.z * other.z )
    def __str__(self):
        return "(" + str(self.x) + "," + str(self.y) + "," + str(self.z) + ")"
    def __mul__(self,other):
        temp = MyVector(0,0,0)
        temp.x =  self.y * other.z - self.z * other.y
        temp.y =  self.z * other.x - self.x * other.z
        temp.z =  self.x * other.y - self.y * other.x
        return temp

v1 = MyVector(1,2,3)
v2 = MyVector(2,4,6)
print(v1*v2)
```

The operator * is “overloaded”. Instead of creating a method “VectorProduct”, it’d better redefine the operator * thanks to the key-word “__mul__”. The operator is used with the class “MyVector” and this way makes the programme more ligible.

Hiding information

- When you use a class, you need to have some pieces of information regarding this class. But you don't need to know all the attributes and the methods of this class. It is sought to hide the complexity of the class.
- When you use a object it is more important to know what services the class offers.

```
v1 = MyVector(1,2,3)
print(v1.Norm())
v1.x  = 34
print(v1.x)
```

The attribute x can be accessed outside the class. The dot notation is used to access to the attribute.

```
========================= RESTART:
3.7416573867739413
34
```

Python allows you "naturally" to display the content of the attributes and modify the value of these same attributes. It even allows you to create other attributes. In fact Python is quite bad at hiding data.

```
class MyVector(object):
    def __init__(self,x,y,z):
        self.__x = x
        self.__y = y
        self.__z = z
    def Norm(self):
        return ( self.__x * self.__x + self.__y * self.__y + self.__z * self.__z )**0.5

myObject = MyVector(1,2,3)
print (myObject.Norm())
print (myObject.__x)
```

The three attributes making up the vector can be hidden from the user of the class "MyVector". For that, the attributes must be suffixed with two underscores.
It is said then that the attributes have a "private" status and can but be accessible inside the class

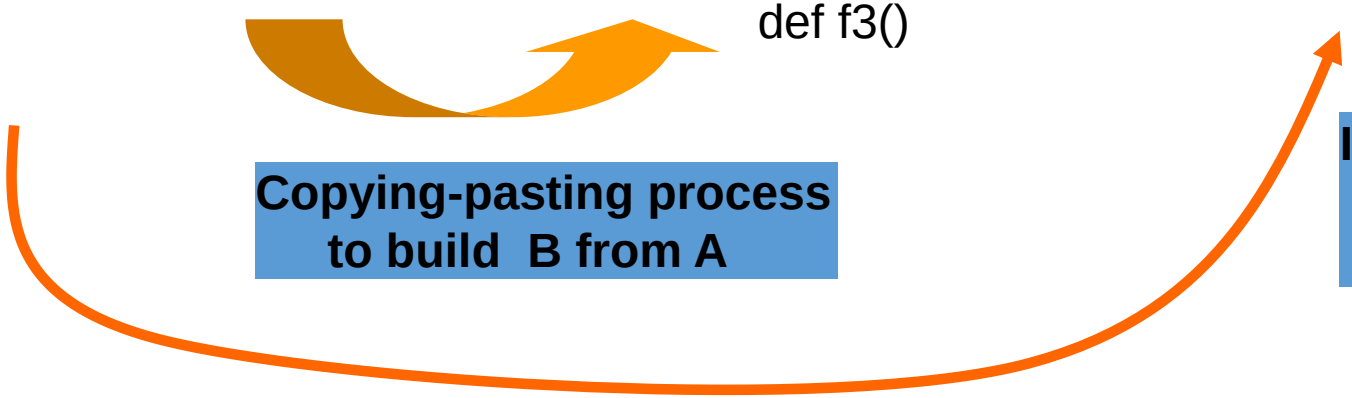
```
== RESTART: C:\Users\sbouter\Travail\langage_python\prepa_cours\theClasses.
py ==
3.7416573867739413
Traceback (most recent call last):
  File "C:\Users\sbouter\Travail\langage_python\prepa_cours\theClasses.py",
line 135, in <module>
    print (myObject.__x)
AttributeError: 'MyVector' object has no attribute '__x'
>>>
```

The inheritance: the approach of the beginner in programming

```
class A:
        x1 = 10
        x2 = 0.0
        def f1()
        def f2()
```

```
class B:
        x1 = 10
        x2 = 0.0;
        x3 ="zZz"

        def f1()
        def f2()
        def f3()
```

```
class B( A) {
        x3 ="zZz"


        def f3();
```

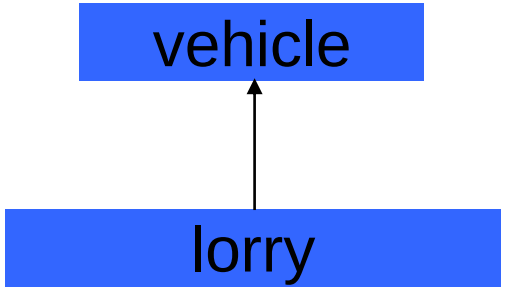


Copying-pasting process to build B from A

Inheritance mechanism: class B inherits the attibutes and the methods of the class A

112

Inheritance



Programing an application requires to get all the attributes and the methods which are already existing in another class. So a class can inherit the attributes and the methods from another class. The inheritance mechanism enables to define a new class from an existing class

Then it is possible to add new attributes and new methods in order to specialise this class. As a matter of fact, the new class doesn't present any fundamental differences.

When a class "lorry" inherits from a class "vehicle", the class "lorry" is called the derived class or subclass and the class "vehicle", the base class or superclass.

Class “vehicle”, the base class

```
class vehicle(object):
    ident = 0
    brand = ""
    horsePowerTax = 0
    power = 0.0
    def __init__(self):
        self.dataInput()

    def dataInput(self):
        #print("\nInput the ID =")
        self.ident =  int(input("Input the ID ="))
        self.brand =  input("Input the brand =")
        self.horsePowerTax =  int(input("Input the horse Power Tax ="))
        self.power =  float(input("Input the Power ="))

    def powerDisplaying(self):
        if self.power >110:
            print("excessive consumption possible")
        else:
            print("fair gas consumption");

    def taxDisplaying(self):
        if self.horsePowerTax >7 and self.power <110:
            print("too much tax for the power")
        else:
            print("fair tax")
```

The attributes of the class “vehicle”

The constructor is intended to run the entering the data

The specialised class, “Lorry”

```
class lorry(vehicle):
    load = 0
    nb_axle = 0;
    def __init__(self):
        vehicle.__init__(self)
        self.Lsaisir()

    def Lsaisir(self):
        self.load = int(input("Enter the load ="))
        self.nb_axle =  int(input("Enter the axle nb ="))

    def display_load(self):
        if self.load >30000:
            print("high consumption possible and careful driving")
        else:
            print("fair gas consumption for the load")

    def display_axle(self):
        if  self.nb_axle >3 and self.load < 30000:
            print("high gas consumption for the load and the nb of axle")
        else:
            print("all is OK for the load and the nb of axle")
```

new attributes

Calling the constructor of the base class needed

new methods

Instantiation of the derived class

v1 “lorry” class object

```
v1 = lorry()
v1.powerDisplaying()
v1.taxDisplaying()
v1.display_load()
v1.display_axle()
```

Calling “vehicle” class methods

Calling “lorry” class methods

```
= RESTART: C:/Users/sbouter/Travail/langa
Input the ID =4
Input the brand =sdfg
Input the horse Power Tax =300
Input the Power =456
Enter the load =30000
Enter the axle nb =4
excessive consumption possible
fair tax
fair gas consumption for the load
all is OK for the load and the nb of axle
>>>
```

“vehicle” class constructor

“lorry” class constructor

Class variables

- Class variables called static member attributes and in the class. The term “class variables” fosters the fact they belong to the class itself, rather than to instances of that class.
- A class variable of a class is shared by each instance created during the application running

```
class vehicle(object):
    ident = 0
    brand = ""
    horsePowerTax = 0
    power = 0.0
    tag = 1
    def __init__(self):
        self.ident = vehicle.tag
        vehicle.tag +=1
        self.dataInput()
    def dataInput(self):
        self.brand = input("Input the brand =")
        self.horsePowerTax = int(input("Input the horse Power Tax ="))
        self.power = float(input("Input the Power ="))
    def identDisplaying(self):
        print("ident = ",self.ident);
    def powerDisplaying(self):
        if self.power >110:
            print("excessive consumption possible")
        else:
            print("fair gas consumption");
    def taxDisplaying(self):
        if self.horsePowerTax >7 and self.power <110:
            print("too much tax for the power")
        else:
            print("fair tax")
```

Class variable “tag” initialized to 1

The variable “tag” is incremented each instantiation

```
Input the brand =AAAA
Input the horse Power Tax =4
Input the Power =56
ident =  1
Input the brand =BBBB
Input the horse Power Tax =7
Input the Power =89
ident =  2
Input the brand =CCCCC
Input the horse Power Tax =9
Input the Power =134
ident =  3
```

```
v1 = vehicle()
v1.identDisplaying()
v2 = vehicle()
v2.identDisplaying()
v3 = vehicle()
v3.identDisplaying()
```

Integrating classes in a module

The threads, their properties

- A thread is not a process. A process owns its own memory space. A thread shares a memory space with the other threads. Indeed these threads work the memory space defined by the process which launches them.

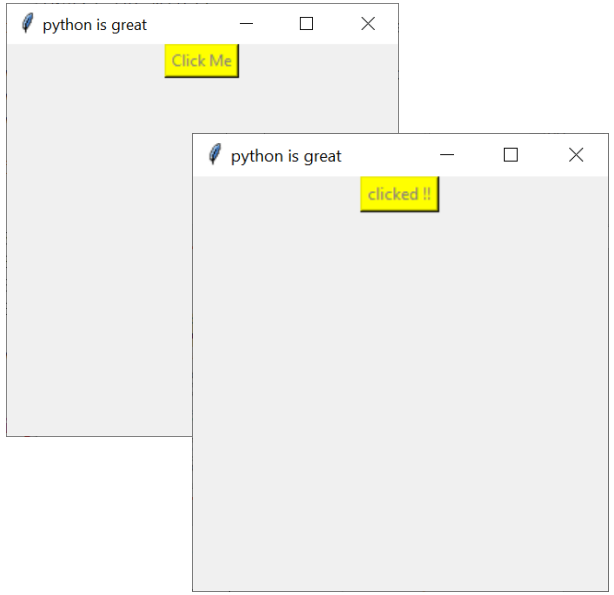
Graphical User Interface: TKinter

- A first simple example
- A second example with a oriented-object programming approach
- Buttons and callback functions
- Slider

First example

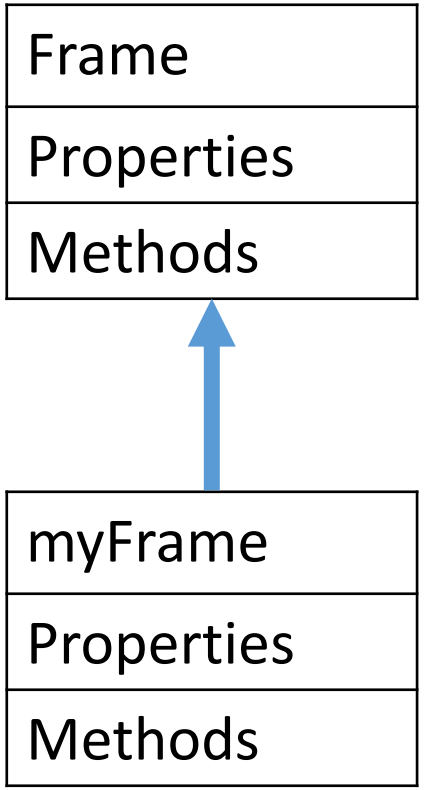
Installing:

- py -m pip install tk




```
from tkinter import *

myWindow = Tk()
myWindow.title("python is great")
myWindow.geometry('300x300')

def clicked():
    myButton['text'] = "clicked !!"

myFrame = Frame(myWindow)
myFrame.pack()
myButton = Button(myFrame, fg = "grey", bg = "yellow", text="Click Me", command = clicked)
myButton.grid(column= 0, row=0)

myWindow.mainloop()
```

Inheriting from the class Frame of Tkinter library



The base class with its base properties and methods

The derived class with additional properties and methods

```
from tkinter import *

class Application(Frame):
    def __init__(self, _theWindow):
        ...

    def create_widgets(self):
        ...
    def say_hi(self):
        ...
    def goodBye(self):
        ...


myWindow = Tk()
app = Application(myWindow)
app.mainloop()
```

How to build the derived class

```
from tkinter import *

class Application(Frame):
    def __init__(self, _theWindow):
        ...

    def create_widgets(self):
        ...
    def say_hi(self):
        ...
    def goodBye(self):
        ...


myWindow = Tk()
app = Application(myWindow)
app.mainloop()
```

The class “Application inherits the methods and the properties of class “Frame”

Passing the reference of the window which has been instantiated

The constructor

Creating an object “window”

Example of a class inheriting from the class “Frame”

```
from tkinter import *

class Application(Frame):
    def __init__(self, _theWindow):

        self.theWindow = _theWindow
        super().__init__()
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.hi_there = Button(self)
        self.hi_there["text"] = "Hello World\n(click me)"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")

        self.quit = Button(self, text="QUIT", fg="red",command= self.goodBye)
        self.quit.pack(side="bottom")

    def say_hi(self):
        print("hi there, everyone!")
    def goodBye(self):
        print("Bye!")
        self.theWindow.destroy()

myWindow = Tk()
app = Application(myWindow)
app.mainloop()
```

Calling the base class constructor

“create_widgets()” is intended to create the buttons “Hello…” and “Quit”

The button called “hi_there” is an attribute of the class “Application” which has also attributes “text”, “command”,…

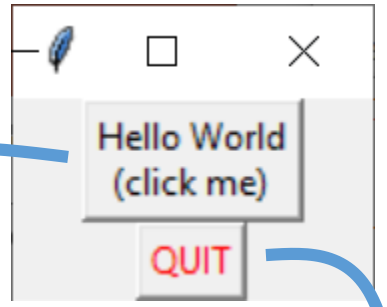
Example a class of inheriting the class “Frame”

```
from tkinter import *

class Application(Frame):
    def __init__(self, _theWindow):

        self.theWindow = _theWindow
        super().__init__()
        self.pack(side = BOTTOM)
        self.create_widgets()

    def create_widgets(self):
        self.hi_there = Button(self)
        self.hi_there["text"] = "Hello World\n(click me)"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")

        self.quit = Button(self, text="QUIT", fg="red",command= self.goodBye)
        self.quit.pack(side="bottom")

    def say_hi(self):
        print("hi there, everyone!")
    def goodBye(self):
        print("Bye!")
        self.theWindow.destroy()

myWindow = Tk()
app = Application(myWindow)
app.mainloop()
```

The callback functions are bound to the buttons “Hello…” and “Quit”. They are called as soon as there is a click on a button.

Defining the Callback functions

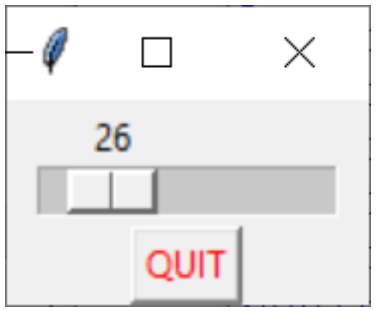
Waiting for signals that will trigger callback functions




```
= RESTART: C:\Users\
Ex.py
hi there, everyone!
hi there, everyone!
hi there, everyone!
hi there, everyone!
hi there, everyone!
Bye!
>>>
```

Other widgets of the library: A silder

```
from tkinter import *

class Application(Frame):
    def __init__(self, _theWindow):

        self.theWindow = _theWindow
        super().__init__()
        self.pack( side = BOTTOM)
        self.create_widgets()

    def create_widgets(self):
        mySlider = Scale(self.theWindow, from_ =0, to=200, orient=HORIZONTAL, command = self.giveValue)
        mySlider.set(100)
        mySlider.pack(side=TOP)

        self.quit = Button(self, text="QUIT", fg="red",command= self.goodBye)
        self.quit.pack(side=BOTTOM)

    def giveValue(self,pos):
        print("position = ",pos)
    def goodBye(self):
        print("Bye!")
        self.theWindow.destroy()

myWindow = Tk()
app = Application(myWindow)
app.mainloop()
```

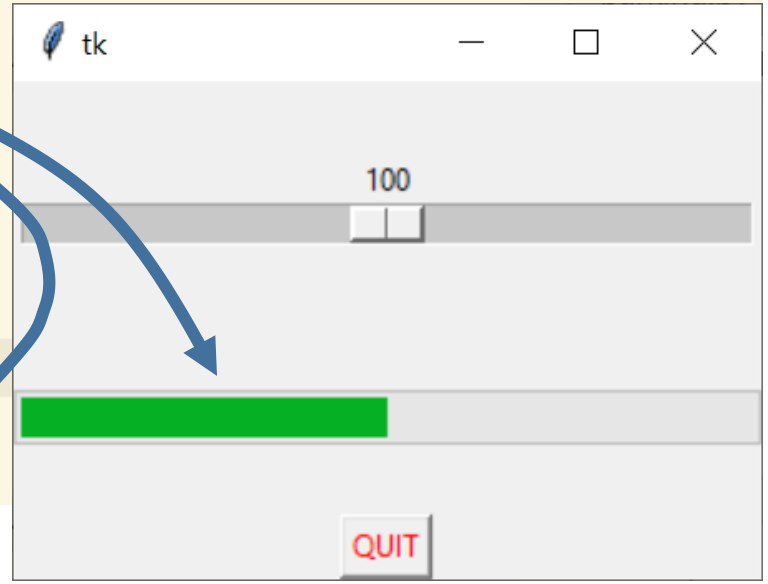



```
position =  56
position =  53
position =  50
position =  47
position =  44
position =  41
position =  38
Bye!
>>>
```

The callback function of the class “Scale” has an mandatory input parameter, the position of the cursor, here “pos”

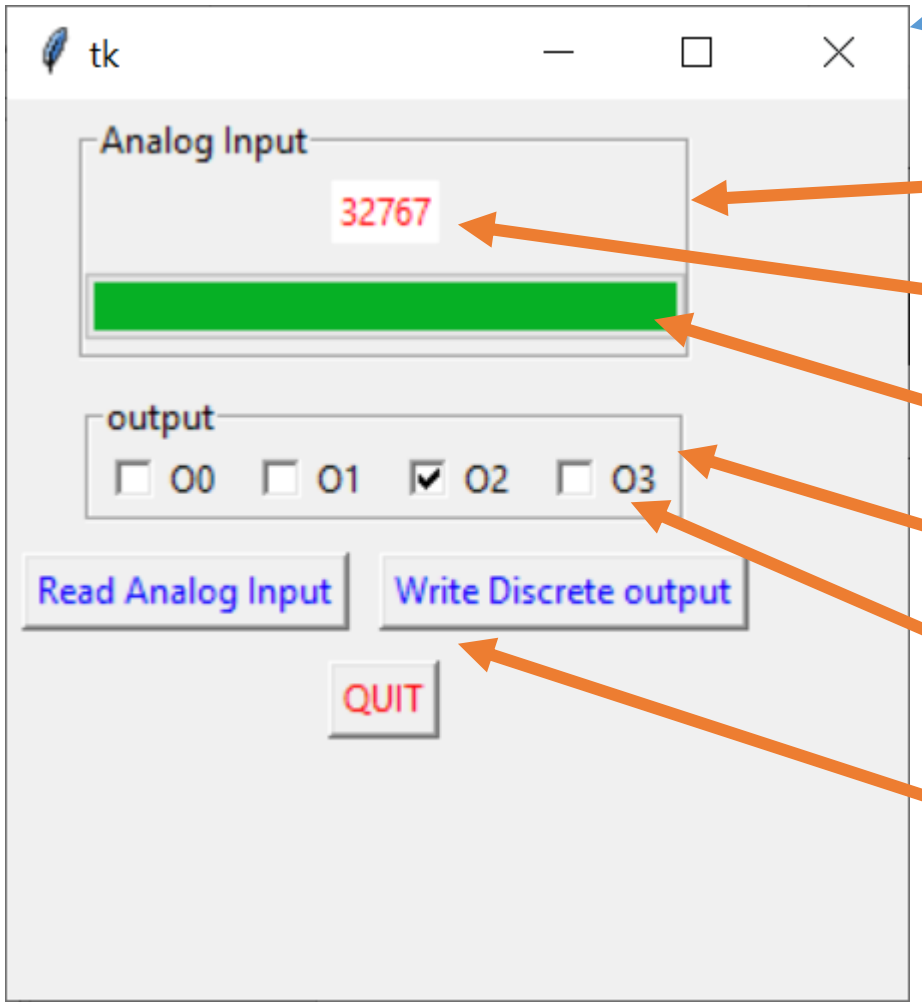
ProgressBar

```
def create_widgets(self):
    mySlider = Scale(self.theWindow, from_ =0, to=200, orient=HORIZONTAL, command = self.giveValue)
    mySlider.set(100)
    mySlider.pack(side=TOP, expand = True, fill ="x")
    self.myProgressBar = ttk.Progressbar(self.theWindow,orient='horizontal',mode='determinate',length=200)
    self.myProgressBar.pack(side=TOP, expand = True, fill ="x")
    self.myProgressBar["maximum"] = 200
    self.myProgressBar["value"] = 0

    self.quit = Button(self, text="QUIT", fg="red",command= self.goodBye)
    self.quit.pack(side=BOTTOM, expand = True)
def giveValue(self,pos):
    self.myProgressBar["value"] =pos
    print("position = ",pos)
```



self.myProgressBar is an attribute of the class.
Its scope covers the class

Example of a Dialog box



The main window, a frame.

A first “Labelframe” belonging to the main window

A label intended for text, with white backgroung and red foreground integrated into the first LabelFrame

A “ProgressBar” intended to visualise graphically values, integrated into the first LabelFrame

A second “Labelframe” belonging to the main window

“Checkboxes” intended to visualise graphically boolean values, integrated into the second LabelFrame

Buttons integrated into the main window. They are usually associated to event such as “click on button”

How to structure the Dialog box: the grid

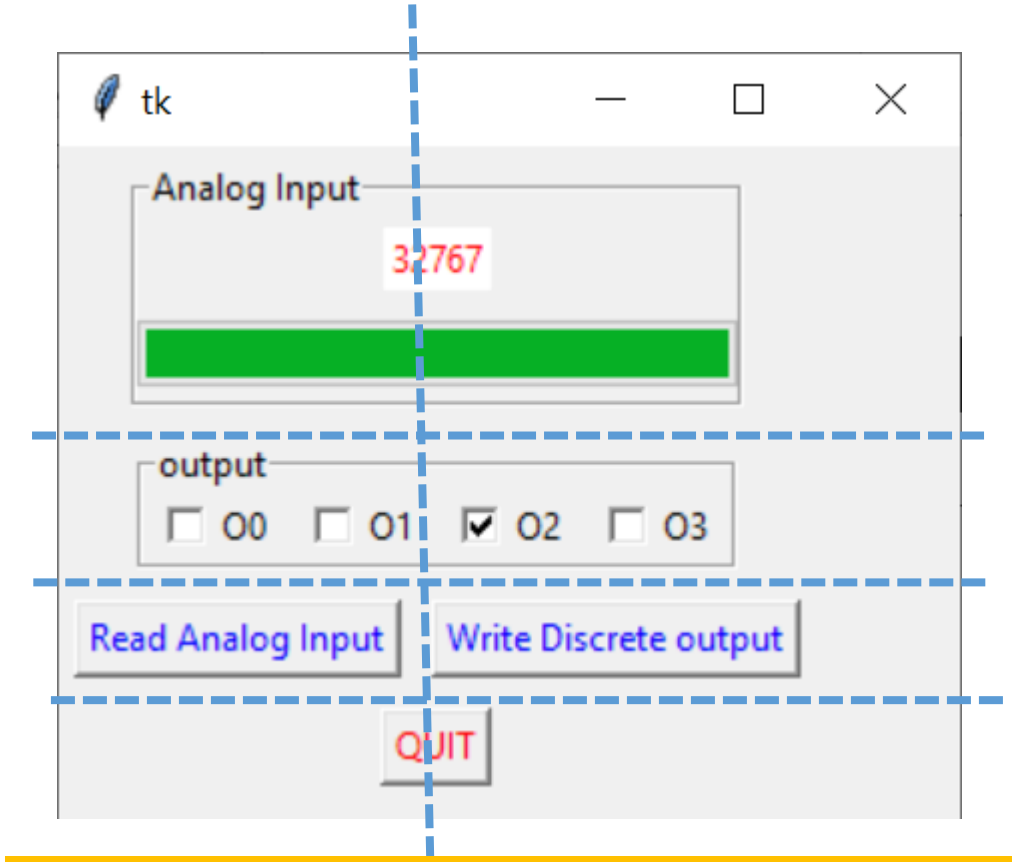
cell

column

row

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

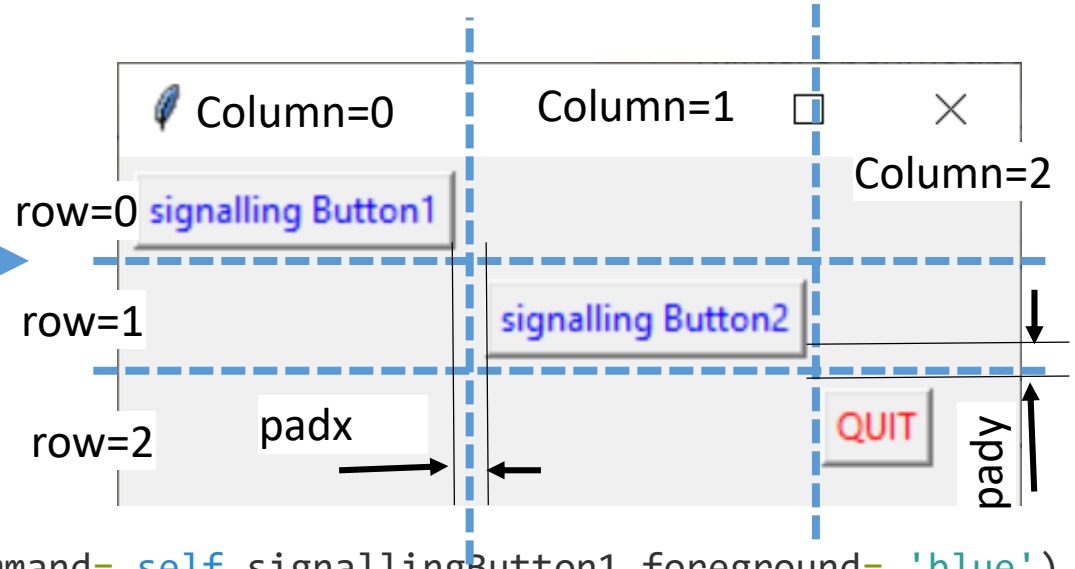
(row, column)



4 rows and 2 columns), some graphical elements span 2 columns

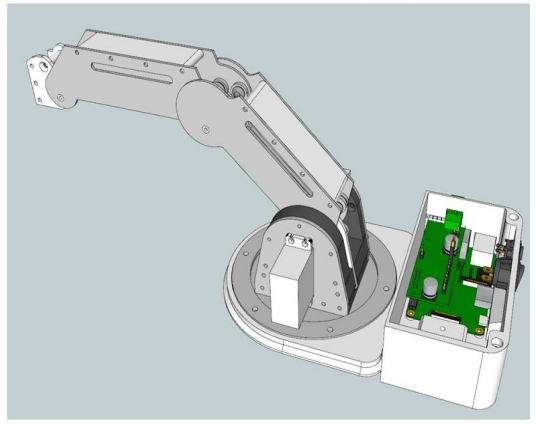
Organising the layout of a Dialog box

The location of the buttons

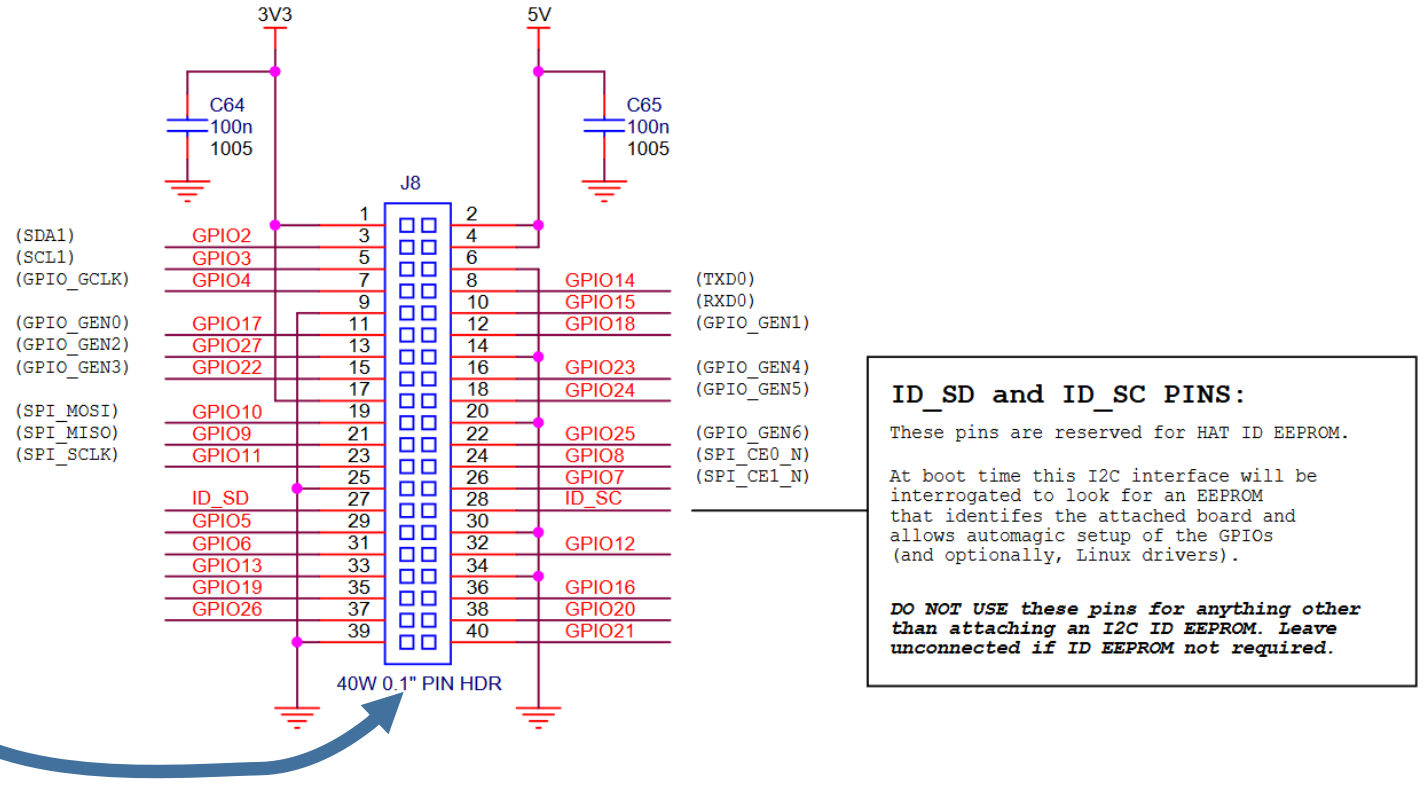



```
class Application(tk.Tk):
    def __init__(self):
        super().__init__()
        self.geometry("300x200")
        self.create_widgets()

    def create_widgets(self):
        self.button1 = tk.Button(text="signalling Button1", command= self.signallingButton1,foreground= 'blue')
        self.button1.grid(row = 0,column=0, pady= 5, padx = 5 )

        self.button2 = tk.Button(text="signalling Button2",command= self.signallingButton2,foreground= 'blue')
        self.button2.grid(row = 1, column=1, pady= 5, padx = 5)

        self.quit = tk.Button(text="QUIT", command= self.goodBye, foreground= 'red')
        self.quit.grid(row = 2, column=2,                  pady= 5)

app = Application()
app.mainloop()
```

Embedded System:Raspberry board



The Raspberry board



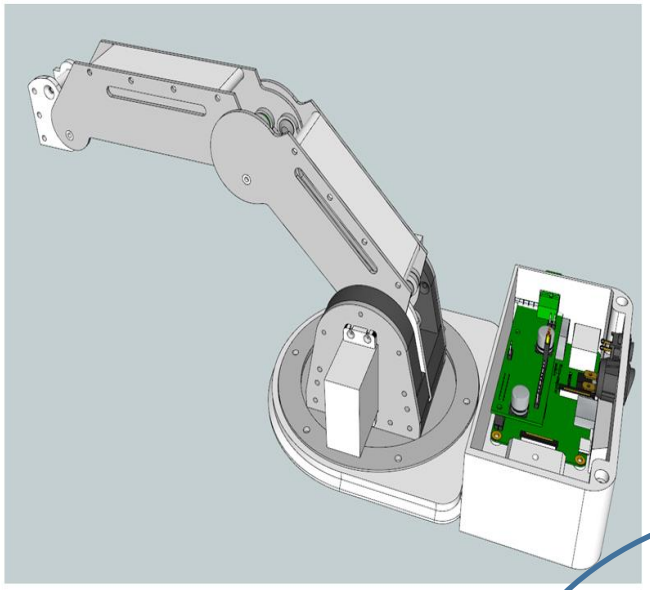
ID_SD and ID_SC PINS:

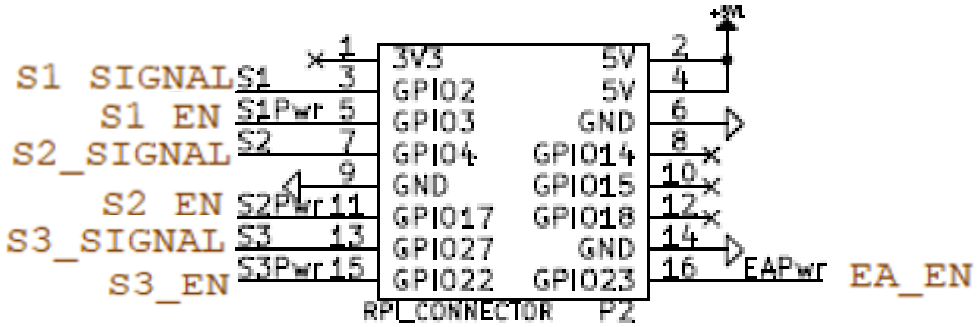
These pins are reserved for HAT ID EEPROM.

At boot time this I2C interface will be interrogated to look for an EEPROM that identifes the attached board and allows automagic setup of the GPIOs (and optionally, Linux drivers).

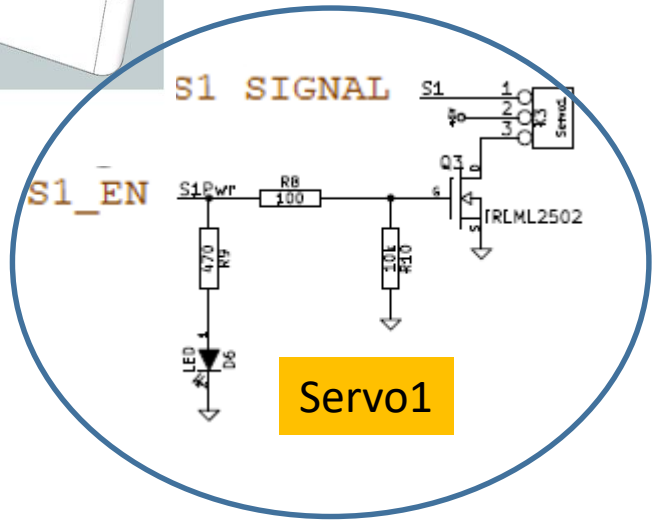
DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.

Pins used for the project of controlling a robot arm

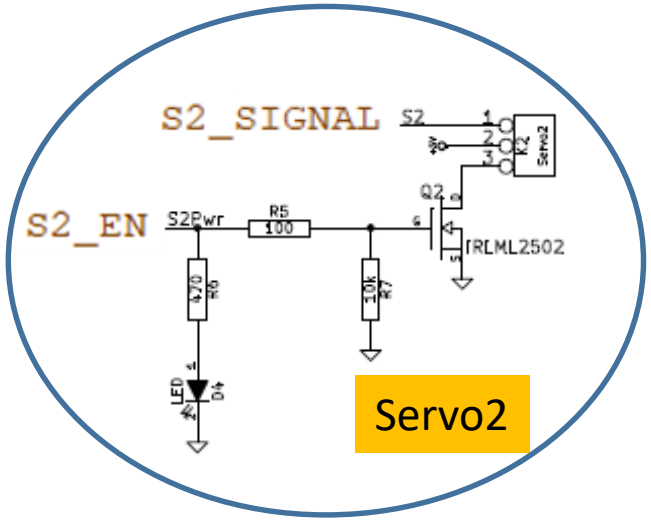




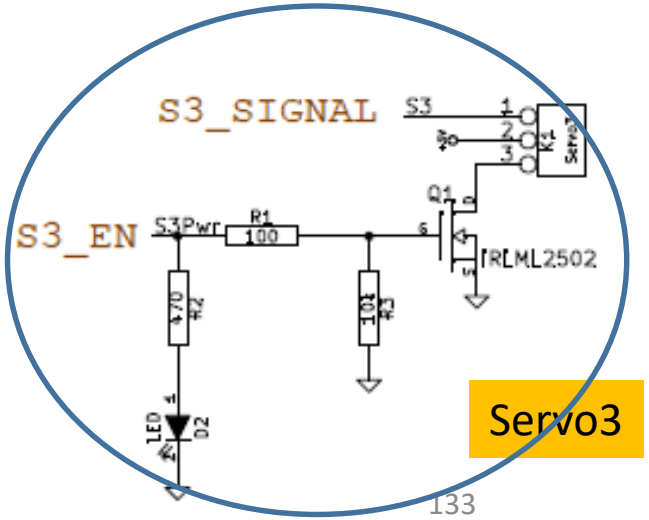
Electro-magnet control



Servo1



Servo2



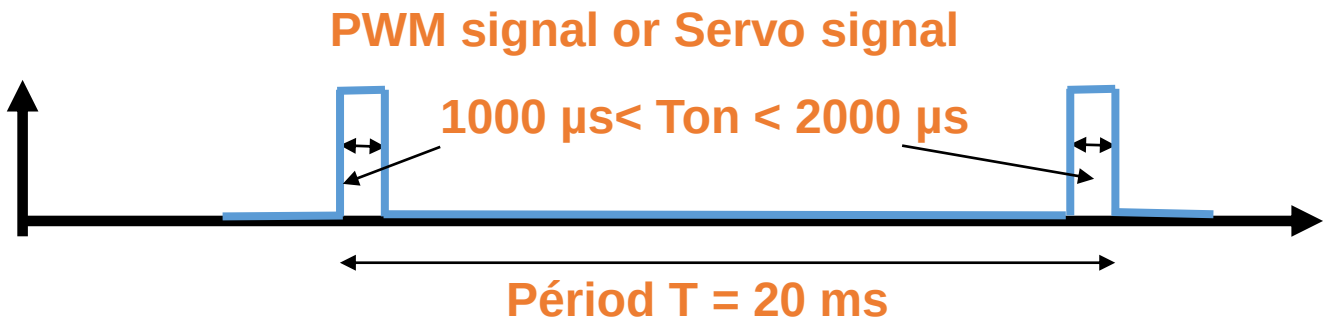
Servo3

Labels, Number and purposes of the Input/Output

actuator	Label	pin	purpose
Servo1	S1_EN	GPIO3	Enable Servo controlling
Servo1	S1_SIGNAL	GPIO2	Signal Servo between 1000 µs and 2000 µs, period = 20 ms
Servo2	S2_EN	GPIO17	Enable Servo controlling
Servo2	S2_SIGNAL	GPIO4	Signal Servo between 1000 µs and 2000 µs, period = 20 ms
Servo3	S3_EN	GPIO22	Enable Servo controlling
Servo3	S3_SIGNAL	GPIO27	Signal Servo between 1000 µs and 2000 µs, period = 20 ms
Electro-magnet	EA_EN	GPIO23	Switch on/off the electro-magnet

#define directives needed to run the programme

```
#define S1_EN          3
#define S2_EN         17
#define S3_EN         22
#define EA_EN         23
#define S1_SIGNAL      2
#define S2_SIGNAL      4
#define S3_SIGNAL     27
```



Input/Output accessing library : pigpio

apt-get is a software tool on the command line intended to you to install and uninstall packages from an APT repository.

Checking the library:
$ pigpiod -v

Installing the library:
$ sudo apt-get update
$ sudo apt-get install pigpio
$ pigpiod -v

1° line: **getting** information on an updated version of packages or their dependencies.
2° line: installing the library
3° line: checking the version

Daemon pigpio

- it is a pigpio library utility which runs as a daemon. The pigpio library runs in the background accepting commands from the pipe and socket interfaces.
  - The pipes are used to connect processes together so that the output of one becomes the input of the other.
  - The sockets is another way which allows the processes to communicate.
- The pigpiod utility requires sudo privileges to launch the library but thereafter the pipe and socket commands may be issued by normal users.
- This daemon is used when we have to control the inputs/outputs from the shell

Input/output accessing command via a deamon

- Using command controlling the input/output of the board requires that that pigpio daemon be running : **sudo pigpiod** (d for daemon)

```
echo modes 4 w > /dev/pigpio # set gpio4 as output (write)
echo modes 17 w  > /dev/pigpio # set gpio17 as output (write)
echo modes 23 r  > /dev/pigpio # set gpio23 as input (read)
echo modes 24 0  > /dev/pigpio # set gpio24 as ALT0
#ALT0? this input/output has several functions(up to 8), in this case, it is set as an input

echo servo 4 1500  > /dev/pigpio # start 1500 us servo pulses on gpio4
echo pwm 17 192  >/dev/pigpio # start 75% dutycycle PWM on gpio17 (192/255 = 75%)

echo r 23  > /dev/pigpio # read gpio23
echo w 24 1  > /dev/pigpio # write 1 to gpio 24

echo servo 4 0  > /dev/pigpio # stop servo pulses
echo pwm 17 0  > /dev/pigpio # stop pwm
```

The character # announces a comment so what follows is discarded. It does not be interpreted and run by the shell

Class pi: access to the GPIO

This module encompasses the class “pi” gives access to GPIO. The class is initialised with the method “gpio.pi(host, port, show_errors)”

Parameters:

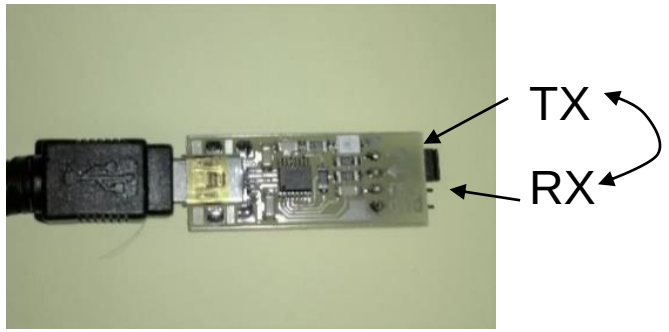
- host:= the host name of the Pi on which the pigpio daemon is running. The default is localhost unless modified thanks to the PIGPIO_ADDR environment variable.
- port:= the port number on which the pigpio daemon is listening. The default is 8888 unless modified thanks to the PIGPIO_PORT environment variable. The pigpio daemon must have been started with the same port number.

Controlling PWM outputs

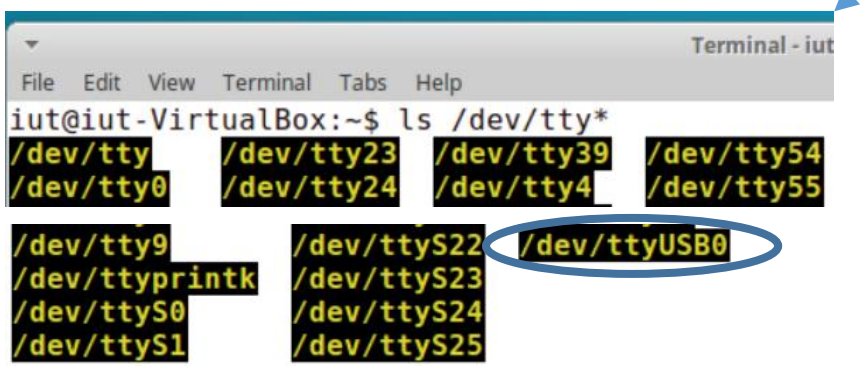
```
#  or specify host, default port(another Raspberry Pi)
pi = pigpio.pi('otherpi')
# or specify host and port (another Raspberry Pi and port number other than 8888)
pi = pigpio.pi('otherpi', 7777)
```

```
if not pi.connected: # this test exits script if connection has failed
    exit()
pi.set_mode(17, 1)
pi.write(17, True)
time.sleep(1) # waiting for 3 s
pi.set_servo_pulsewidth(4, 0) # off
time.sleep(3) # waiting for 3 s
pi.set_servo_pulsewidth(4, 1000) # safe anti-clockwise
time.sleep(3) # waiting for 3 s
pi.set_servo_pulsewidth(4, 1500) # centre
time.sleep(3) # waiting for 3 s
pi.set_servo_pulsewidth(4, 2000) # safe clockwise
time.sleep(3) # waiting for 3 s
pi.set_servo_pulsewidth(4, 0) # off
pi.set(17, False)
pi.stop() # release the pigpio ressources
```

Asynchronous serial link: example of connection for a first exercise



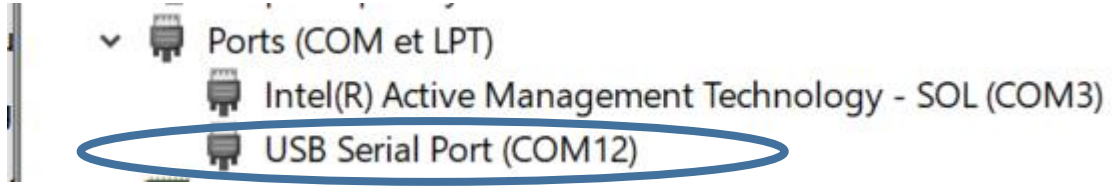
The pins RX and TX are connected to each other



With Linux, when you have plugged thus board to the USB connector of the PI board, check the port which appears by running the following command: $ ls /dev/tty*.

A virtual port is defined over USB

With Windows, check the port which appears by looking into the peripheral manager



Pyserial a library dealing with the serial port

```
import time #the needed libraries
import serial

sendCommand = bytearray(6)
receivedData = bytearray(6)

sendCommand[0] = ord('R')
sendCommand[1] = 65
sendCommand[2] = 66
sendCommand[3] = 67
sendCommand[4] = 68
sendCommand[5] = ord('\n')

# the programme is opening a virtual port defined over USB
h1 = serial.Serial("COM12",baudrate = 9600,timeout = None) # no timeout

test = 0
while test != 'F':
    h1.write(sendCommand) #each character is stored into a byte
    receivedData = h1.readline()
    print("data received",receivedData)
    print("enter F to finish")
    test = input()
h1.close()
```

Opening the selected port. This function return an abstract reference to a resource, here the port ttyUSB0, called an handle

Sending a message

Waiting for data

Closing the port

```
= RESTART: C:/Users/sbouter
LinkSimpleExamplePySerialLi
data received b'RABCD\n'
enter F to finish
A
data received b'RABCD\n'
enter F to finish
Z
data received b'RABCD\n'
enter F to finish
E
data received b'RABCD\n'
enter F to finish
Y
data received b'RABCD\n'
enter F to finish
F
>>>
```

Asynchronous serial link with the pigpio library

```
import time #the needed libraries
import pigpio
pi = pigpio.pi()#give access to GPIO, use defaults

if not pi.connected: # this test exits script if connection has failed
    exit()
h1 = pi.serial_open("/dev/ttyUSB0",9600,0)
if not h1 >= 0:# this test exits script if opening port has failed
    exit()
pi.serial_write(h1,"hello World")

nbOfBytesReceived = 0
while nbOfBytesReceived == 0:
    nbOfBytesReceived = pi.serial_data_available(h1)
(nbOfBytesReceived,ArrayBytesReceived) = pi.serial_read(h1,60)
if nbOfBytesReceived > 0:
    print("\nMessage Received = " + str(ArrayBytesReceived,'utf-8'))
pi.serial_close(h1)
pi.stop() # release the pigpio ressources
```

Opening the selected port. This function return an abstract reference to a resource, here the port ttyUSB0, called an handle

If the port is opened the handle is greater than or equal to 0

Sending a message

Waiting for data

Reading data, at most 60 bytes. The number of bytes is greater than or equal to 60, but only 60 bytes will be stored. This function returns the number of bytes received and an array containing the bytes received

Closing the port

Environment Python: Notebook Jupyter

Jupyter

- Kernels : Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel and any output is returned back to the cell to be displayed. The kernel's state persists over time and between cells — it pertains to the document as a whole and not individual cells.

- For example, if you import libraries or declare variables in one cell, they will be available in another. In this way, you can think of a notebook document as being somewhat comparable to a script file, except that it is multimedia. Let's try this out to get a feel for it. First, we'll import a Python package and define a function.
- You cannot rename a notebook while it is running, so you've first got to shut it down. The easiest way to do this is to select "File > Close and Halt" from the notebook menu. However, you can also shutdown the kernel either by going to "Kernel > Shutdown" from within the notebook app or by selecting the notebook in the dashboard and clicking "Shutdown" (see image below).
- Save and Checkpoint: Now we've got started, it's best practice to save regularly. Pressing Ctrl + S will save your notebook by calling the "Save and Checkpoint" command, but what this checkpoint thing? Every time you create a new notebook, a checkpoint file is created as well as your notebook file; it will be located within a hidden subdirectory of your save location called .ipynb_checkpoints and is also a .ipynb file. By default, Jupyter will autosave your notebook every 120 seconds to this checkpoint file without altering your primary notebook file. When you "Save and Checkpoint," both the notebook and checkpoint files are updated. Hence, the checkpoint enables you to recover your unsaved work in the event of an unexpected issue. You can revert to the checkpoint from the menu via "File > Revert to Checkpoint."

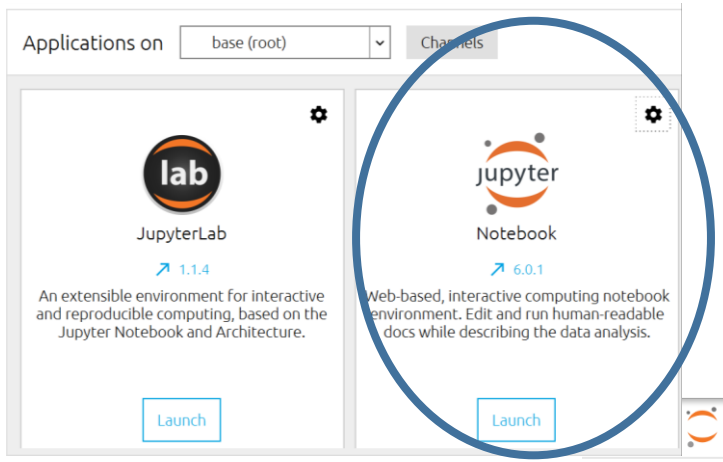
Installing basemap

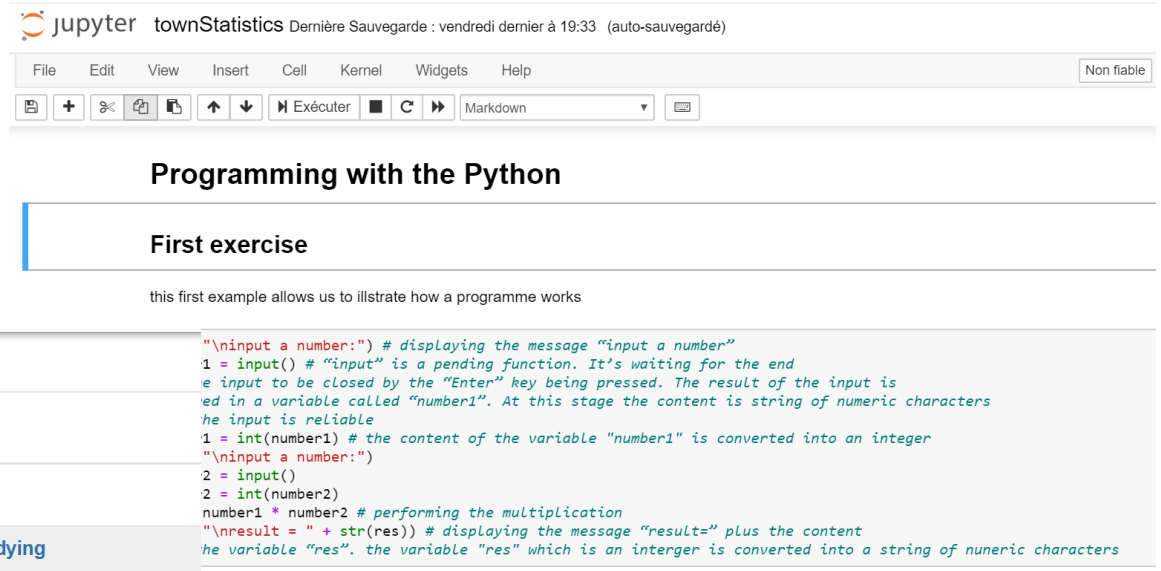
```
(base) C:\Users\sbouter>conda install numpy matplotlib basemap
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\ProgramData\Anaconda3

  added / updated specs:
    - basemap
    - matplotlib
    - numpy
```

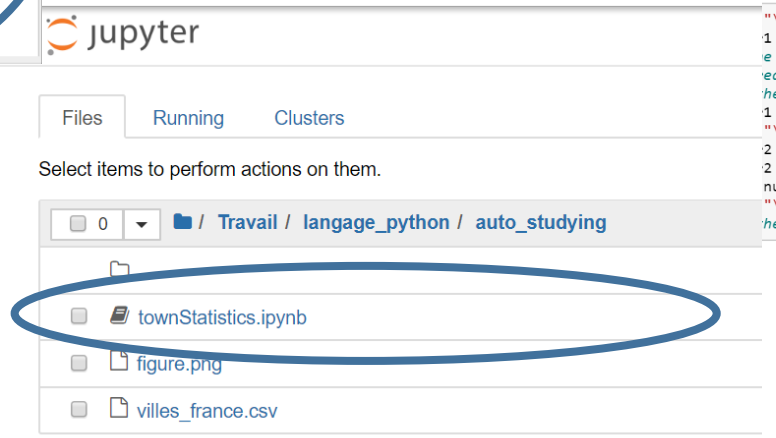
```
The following NEW packages will be INSTALLED:
  basemap          pkgs/main/win-64::basemap-1.2.0-py37h4e5d7af_0
  geos             pkgs/main/win-64::geos-3.6.2-h9ef7328_2
  proj4            pkgs/main/win-64::proj4-5.2.0-ha925a31_1
  pyproj           pkgs/main/win-64::pyproj-1.9.6-py37h6782396_0
  pyshp            pkgs/main/noarch::pyshp-2.1.0-py_0

The following packages will be UPDATED:

  conda                                4.7.12-py37_0 --> 4.8.0-py37_1

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\sbouter>
```

In order to install libraries or packages it is needed to run Anaconconda as administrator

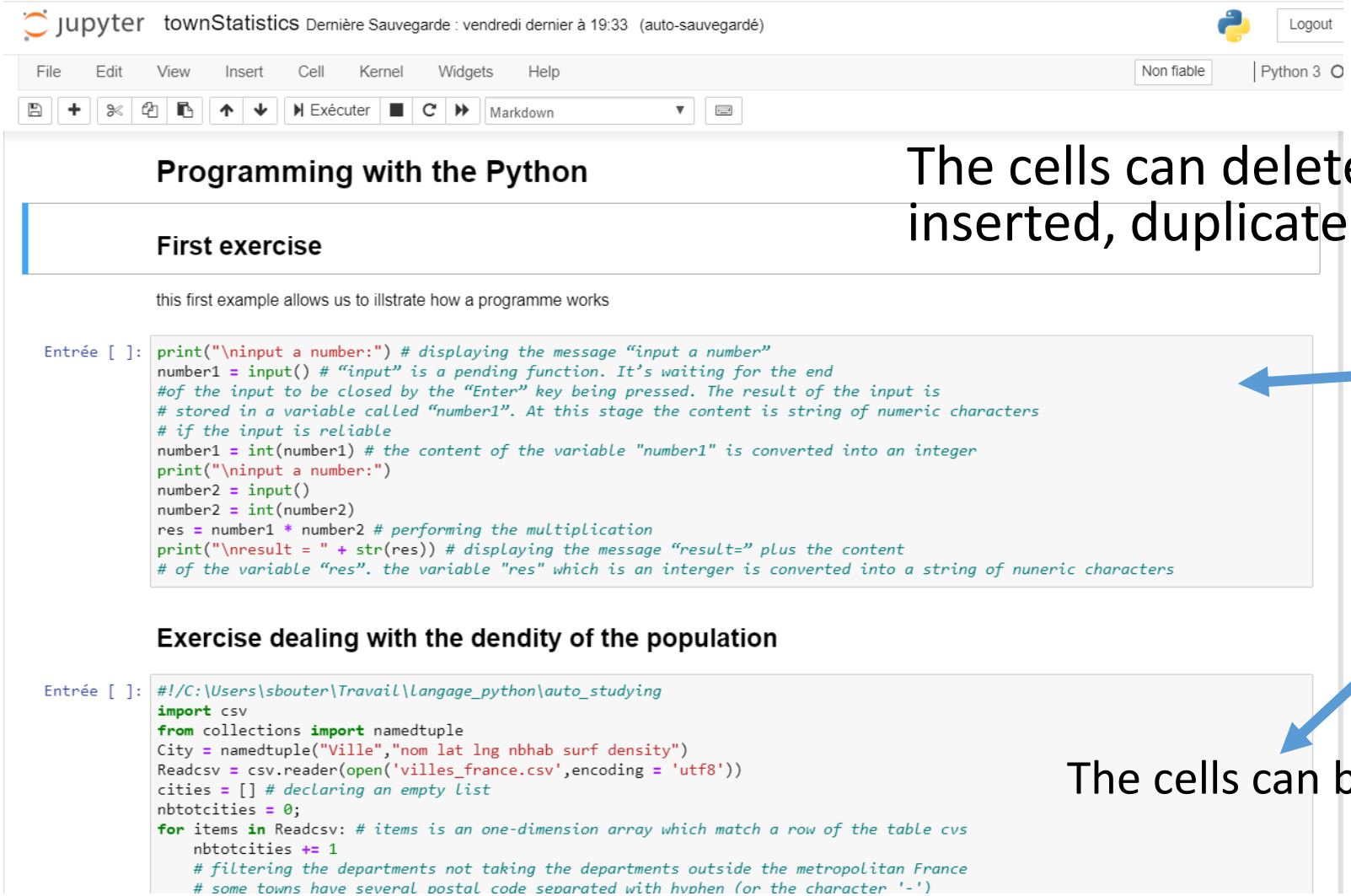
Jupyter Environment







Interface of Jupyter

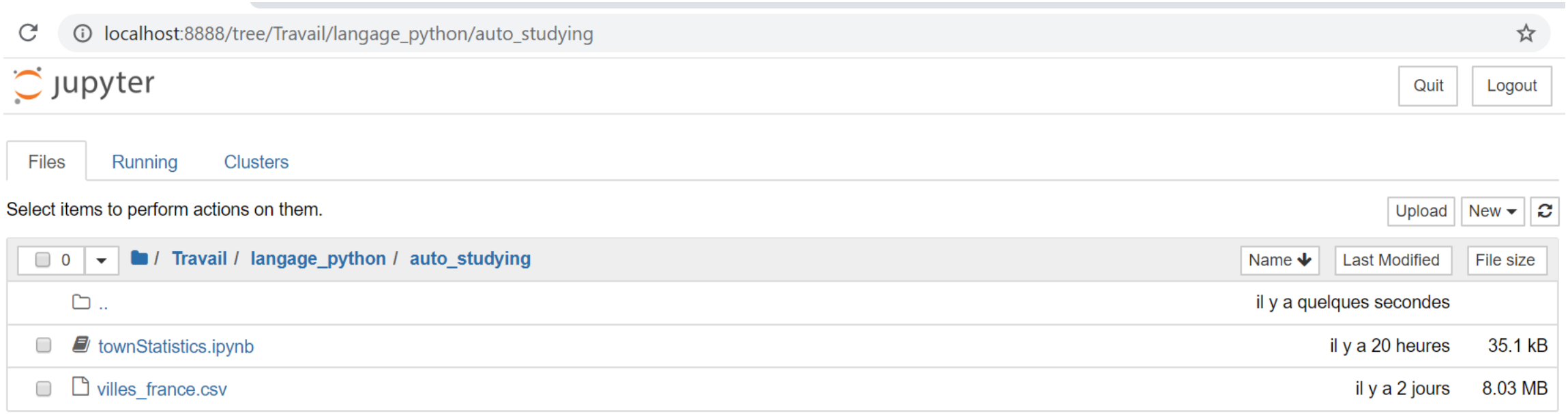


The cells can deleted, cut, inserted, duplicated…

cells

The cells can be run…

Work space


localhost:8888/tree/Travail/langage_python/auto_studying
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/ Travail / langage_python / auto_studying
Name
Last Modified
File size
..
il y a quelques secondes
townStatistics.ipynb
il y a 20 heures
35.1 kB
villes_france.csv
il y a 2 jours
8.03 MB

localhost:8888/notebooks/Travail/langage_python/auto_studying/townStatistics.ipynb

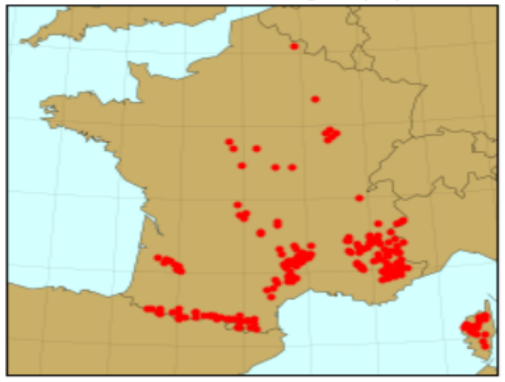
jupyter townStatistics Dernière Sauvegarde : il y a une minute (modifié) Logout

File Edit View Insert Cell Kernel Widgets Help Fiable Python 3

Exécuter Code

```
Ville(nom='Villiers-le-Duc', lat=47.8167, lng=4.71667, nbhab=84.34, surf=100, density=1.1856770215793218),
Ville(nom='Prads-Haute-Bléone', lat=44.22, lng=6.44334, nbhab=165.64, surf=200, density=1.2074378169524271),
Ville(nom='Gavarnie', lat=42.7333, lng=-0.008333, nbhab=82.54, surf=100, density=1.21153380179307)]
```

towns with small density of population



The cells can deleted, cut, inserted, duplicated…

The cells can be run…

```
Entrée [ ]:
```

Controlling a industrial robot

- https://www.universal-robots.com/download/?option=18940