# Contents

## 1 Introduction to Python

1 Spyder environment ...................................................... 3
   1.1 Personal folder organization ..................................... 3
   1.2 Installation of Spyder (Python 3.xx) ............................ 4
   1.3 Windows layout ..................................................... 6
   1.4 Using the Console : interactive mode ............................ 8
   1.5 Using the Editor .................................................... 10
   1.6 More about Spyder .................................................. 12

## 2 Starting with Python .................................................. 16

2.1 Basic programming .................................................... 16
   2.1.1 Operations and types ......................................... 16
   2.1.2 Conditional instructions ..................................... 19
   2.1.3 Loops ............................................................. 20
   2.2 Functions ........................................................... 23
   2.2.1 Definition and execution of functions ...................... 23
   2.2.2 Recursive function .......................................... 25
   2.2.3 Local and Global variables .................................. 26
   2.2.4 Lambda functions ............................................ 27
   2.2.5 The instruction test: if \_name\_ == \_main\_: .............. 27
   2.3 Files ............................................................... 28
   2.4 Overview on Modules and Libraries .............................. 30
   2.5 Some structured types in Python ................................ 34
   2.5.1 String .......................................................... 34
   2.5.2 List ............................................................. 36
   2.5.3 Tuple ........................................................... 39
   2.5.4 Dictionary ..................................................... 41
   2.6 Class ............................................................... 43

## 3 Linear algebra with numpy ........................................... 47

3.1 Vectors and matrices ................................................ 47
3.2 Concatenation of matrices ......................................... 49
3.3 Operations on matrices .............................................. 49
3.4 Complex matrices .................................................... 50
3.5 Copy and Hard copy ................................................ 50
3.6 Display of an array ................................................ 51
3.7 Save an array to a text file ...................................... 51
3.8 Determinant and Inverse .......................................... 51
3.9 Resolution of a linear system .................................... 52
3.10 Eigenvalue .......................................................... 52
3.11 Shape, Nan, Inf .................................................... 53
3.12 The class matrix of numpy ........................................ 53
3.13 Exercises .......................................................... 54
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Graphic with matplotlib</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>2D plotting</td>
<td>56</td>
</tr>
<tr>
<td>4.2</td>
<td>MultiFig and subplot</td>
<td>57</td>
</tr>
<tr>
<td>4.3</td>
<td>Working with text</td>
<td>58</td>
</tr>
<tr>
<td>4.4</td>
<td>Drawing a circle - axis equal</td>
<td>59</td>
</tr>
<tr>
<td>4.5</td>
<td>3D curve plotting</td>
<td>60</td>
</tr>
<tr>
<td>4.6</td>
<td>Surface plotting</td>
<td>61</td>
</tr>
<tr>
<td>4.7</td>
<td>Level set</td>
<td>65</td>
</tr>
<tr>
<td>4.8</td>
<td>Vector fields</td>
<td>65</td>
</tr>
<tr>
<td>4.9</td>
<td>Mouse acquisition</td>
<td>66</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction to Python

1 Spyder environment

1.1 Personal folder organization

Working with Python, or any other computer language, requires some organization in the management of your training files, which can be very numerous. Throughout your training you will create many files, these files must be properly named and stored in appropriate folders with meaningful names.

So first, create successively the following folders according to the figures below.

→ In the folder Documents (1), create the folder My Courses (2)
→ In the folder My Courses (2), create the folder Python (3)
→ In the folder Python (3), create the two folders My Working Space (4) and Python Documents (5)
→ In the folder Python Documents (5), create the folder Python Scripts (7)
And finally:

→ From [http://www-ljk.imag.fr/membres/Luc.Biard/#Teaching...](http://www-ljk.imag.fr/membres/Luc.Biard/#Teaching...) download:
  - the pdf file of the course: **IntroPython.pdf**
  - the associated Python script files (as an archive): **PythonScripts.zip**

From the previous points, you should know where to store these documents.

At that point:

- *The folder Python Documents contains all original source documents (pdf files, scilab scripts,...) that you have downloaded.* Do not modify the initial Python scripts!
- *The folder My Working Space will contain all your training documents (your own programs, exercises,...).* You can do what you want here!

I advise you to copy all the script files (from the folder Python Scripts) in your folder My Workspace, for example in a folder named **WScripts** ("W" for working).

Then you can work on these copies, no matter if you make mistakes...

### 1.2 Installation of Spyder (Python 3.xx)

1. Download anaconda for Windows, e.g., from [https://www.anaconda.com/distribution/#download-section](https://www.anaconda.com/distribution/#download-section)
   (a) → click on the button "Windows" so as to get the following menu

   ![Anaconda Installer Menu](image)

   (b) → download the **Python 3.7 version**, typically (according your computer) the "64-Bit Graphical Installer (662 MB)"

2. At that point, you have downloaded the file **Anaconda3-2019.03-Windows-x86_64.exe**
   Run this executable file so as to install Anaconda (Python, Spyder, Jupyter Notebook,...)
3. Create a shortcut of Spyder on your desktop

4. Launch Spyder from the shortcut on your desktop. You will get the following window, which is the “default Spyder layout”

Buttons 1 allow to shift from the IPython console to the History of your instructions.
Buttons 2 allow to shift between different panes: Help, Variable explorer, File explorer, ...
The other buttons will be specified in the next section.

We recommend to use this default Spyder layout, so that the following section 1.3 may be omitted at first reading.
1.3 Windows layout

The layout of Spyder windows can be re-arranged at your convenience. For this purpose, you need to proceed as follows – and do not worry if you make mistakes, it is always possible to come back to the default Spyder layout.

1. You can close the menu associated with a pane by clicking on the cross button.

2. You can detach a pane by clicking on the button and then move it by clicking on the bar. When moving this pane (or window), it can be integrated at a new position in Spyder or as an independent window outside the Spyder main window.

   For example, you can get the following layout in which each pane contains only one menu.

   ![Spyder Editor and IPython console](image)

   You can also get two separate windows, e.g., one for the editor and another one for the IPython console (which means that all the other panes have been closed).

   ![Spyder Editor and IPython console](image)

3. You can reset Spyder to factory defaults with the menu Tools. This will restart Spyder and reset to default settings.
4. You can choose the panes to be displayed and thus reactivate closed panes with the menu View in 7.

5. You can come back to default Spyder layout with the menu View in 7. For this operation, you must not have detached windows. This action is similar to point 3, but it will not restart Spyder.
6. You can save your current layout (e.g., as “MyLayout”) with the menu View in 7.

1.4 Using the Console: interactive mode

The IPython Console is an interpreter (it works as a calculator): each command is executed immediately after the key return (the key enter)

Experiment the following instructions given below (with a [→] at the end of each line) in the IPython console. Notice that the quote “ ’ ” can be modified through a copy and paste from the pdf-file.

```python
In [1]: print("Hello everybody")
Hello everybody

In [2]: a = 2
#assignment: a is the name of a variable which is assigned
#with the value 2

In [3]: a #checking, to be sure ...
Out [3]: 2 #OK!

In [4]: a = a + 3

In [5]: a
Out [5]: 5 #now the value of the variable a is 5

In [6]: b = 6

In [7]: a+b
Out [7]: 11
#the result of the sum is displayed but not assigned in a variable
```
CHAPTER 1. INTRODUCTION TO PYTHON

In [8]: c = a + b
#now, the result of the sum is saved (assigned) to the variable c

In [9]: print('The sum of a and b is',c)
The sum of a and b is 11
#(see also In [1]) and the variable c is replaced by its current value

In [10]: c = c+d; print('c + d = ',c+d)
c = 25
#if you have been attentive, you must have noticed that
#the assignment ‘ c = c + d ’ has been performed twice

In [11]: my_name = 'Luc'
#my_name is a variable assigned with the string 'Luc'

In [12]: print("My first name is",my_name)
My first name is Luc

• “ In [2] ” : means “ Instruction number 2 ”

• Python comments. In Python, any text written on the right of the symbol “ # ” is ignored, and assumed as a comment. Below, we will see another way to write comments in the editor.

• Variable. A variable allows to store information (numerical values, strings, and more complex structures,...).

  – The variable name can only consist of letters (upcase and lowercase letters), digits and the underlined symbol “ _ ”. The variable name can not start with a digit. E.g, age, Age, age_of_Max, year1, this_year_2016, number_of_iterations,... are valid and different variable names (age ≠ Age).

  – Some variable names are reserved for Python language : the Built-in Functions Following is an (incomplete) list of such Python keywords

        and del from none True as elif global nonlocal try
        assert else if not while break except import or with
        class False in pass yield continue finally is raise def
        for lambda return

    See also : https://docs.python.org/2/library/functions.html

• Assignment. The symbol “ = ” is not the usual maths symbol for equality. It represents the assignment. The syntax is:

        variable_name = valeur or variable_name_1 = variable_name_2

In the second case, the value of the variable variable_name_2 is assigned to the variable variable_name_1

        a = 21.85 : the variable a is assigned with the real value 21.85
        x = y : the current value of the variable y is assigned to the variable x

• It is possible to redisplay, modify and relaunch a previous command with the keyboard keys ↑ and ↓

• Remark. The instructions “ c = a + b ” and “ c=a+b ” are equivalent : blanks are not necessary. I suggest always type a blank before and after the equality sign “ = ”, so as to emphasize the assignment


Remark. The semicolon ‘;’ can separate several instructions typed on a same line: see In [10] (instructions 10 above). These commands will be executed sequentially after the For readability it is better to type one command per line

Miscellaneous. Try the following instructions and then check the value of the involved variables

\[
\begin{align*}
a &= 5; \ b &= 7 \\
a, b &= b, a \ # \text{permutation} \\
x &= y &= 3 \ # \text{double affectation}
\end{align*}
\]

1.5 Using the Editor

It is clearly tedious and inefficient to work with the console, particularly when the number of instructions increases. It is then better to write the instructions in an editor, so that you can run, edit, modify and save them in a file that can be reused later.

We will now work with the editor and consider our first program!

By “program”, we mean, a sequence of instructions that will be executed sequentially.

Consider the same sequence of instructions we typed in the IPython console in interactive mode. So, type the following sequence of instructions in the editor and save this file as \textit{f00prog1.py} in your folder \texttt{My Working Space\WScripts}

```python
print("Hello everybody")
a = 2
a = a + 3
b = 6
c = a + b
print('The sum of a and b is', c)

""
Now, several instructions on a same line, separated by semicolons
""
d = 7; c = c+d; print('c + d = ', c+d)

my_name = 'Luc' \ #the variable 'my_name' is a string of characters
print("My first name is", my_name)
```

Comments

- As in the console, any text on the right of the symbol “#” is assumed to be a comment. Therefore, it works for comments no longer than a line.
- In the editor, and more generally in any Python program file (such as \textit{prog1.py}), any area starting and finishing with a three double quote """ is assumed to be a comment.

```python
""
several
lines of comments
...
""
```

If you click on button \textbf{8} the whole program (i.e., the sequence of instructions from line 1 to line 23 will be executed in the IPython console.
• Notice that any text exceeding the Limit line in the editor will cause a line break when printing.

• You can run a sequence of selected instructions in the editor (e.g., sequence from line 8 to line 19 in the example below) with the button \[9\] or with the CTRL \[\text{Esc}\] command.

In the following sections, in order to work with the different script files, you can either copy these programs from this pdf-file in the editor (but, take care of quotes “’”), or load the associated file from the folder “WScripts” in the editor.

Then, you can test each instruction or group of instructions with the button \[9\] or by typing CTRL \[\text{Esc}\]
1.6 More about Spyder

- Loading and running an existing file from the Editor

  With the menu *File*, open the file `f00prog1.py` from your folder `WScripts`.

You should get the following situation: the current directory of the *File explorer* is unchanged.

Then, run this file with the button `8`. This will automatically update the current directory of the *File explorer*.
• Loading and running an existing file from the File explorer
  Use the File explorer to select your folder WScripts

Then select a file in the File explorer and double-click on it in order to load it in the Editor. You should get the following situation. And of course, you can now run this file.

• Current working directory
  The current working directory is the working directory for the IPython console and the current directory for the File Explorer. You can set it up as follows.

In the menu Tools select the item Preferences and then the item Current working directory. You can then select any directory as the current working directory as shown below. After validation, you will have to restart Spyder.
Variable explorer and the IPython console
Select the Variable explorer as shown below and run for example the file `f00prog1.py` with the button 8.

1. This file `f00prog1.py` is executed in the IPython console
2. At the same time, all variables involved in this program `f00prog1.py` are updated in the Variable explorer

Finally, make a right-click in the IPython console (as shown below) and click on the menu Remove all variables and see what happens in the Variable explorer
• Graphic display
Type and run the following two instructions in the *IPython console* (as shown in the figure below).

```
In [1]: import matplotlib.pyplot as plt
In [2]: plt.plot([1,2,3,4], [1,9,4,16], 'or--')
```

The graphic is displayed directly inside the *IPython console*.

We now indicate the way to configure the Spyder environment in order to obtain graphics in a separate window.

1. Select the menu *Tools/Preferences/IPython console*
2. Select *Graphics* on the right side of the window menu
3. in *Graphics backend* choose *Automatic* (if not)
4. re-launch Spyder
2 Starting with Python

https://openclassrooms.com/courses/apprenez-a-programmer-en-python

2.1 Basic programming

2.1.1 Operations and types

Proceed with file f01Preliminaries.py, in which we consider the following items.

- Basic operations: Addition, subtraction, multiplication, division: +, -, *, /, **
CHAPTER 1. INTRODUCTION TO PYTHON

```python
x = 7
y = 5
result = 2*x - x*y / 2
print(result)  # to check the result of the previous instruction
result = x**2  # x power 2
result -= 7    # equivalent to: result = result - 7
result  # another way to check the result of the previous instruction

the_temperature = -5  # not a Myanmar temperature...
the_temperature += 17
print('The temperature is now', the_temperature, 'degrees Celsius')
```

- Integer division and modulo

```python
"""
Integer division and modulo
"""

```
div = 23 // 5
print(div)

remainder = 23 % 5
print(remainder)

```

- Basic data types in Python

```python
"""
Basic types in Python
"""

# basic types in Python are
# int (integers)
# float (floating numbers for encoding (some) real numbers)
# string (of characters)
# bool (boolean) : True; False

# concatenation of strings
x = "Hello"
y = ' John'
z = ", how are you ?"
u = x+y+z
print(u)

"""
Some roundings
"""

```
round(3.7)  # rounds to the closest integer
round(3.49)  # 3
round(3.5)  # 4
round(-4.65)  # -5
int(-4.65)  # -4
int(2.764)  # 2
```

"""
```python
z = float('inf')
z  # inf
z + z  # inf
z + 3  # inf
5 * z  # inf
y = -z
y   # -inf
z + y  # nan

x = float('NaN')
x  # nan
x + z  # nan
```

- Some useful instructions

```python
Some useful commands in Python

# help
# it is better to type this command directly in the console
help(print)
help(int)
help(bool)

# type
a = 2
type(a)
b = 3.0
type(b)
c = 'Hello sir'
type(c)
type(56)
type(-1.3)
type(' ')  # string of 3 blanks
type(print)  # builtin_function or method

# some new types that will be presented later
x = [2,7]
type(x)
y = (2,7)
type(y)
z = {}
type(z)

# print
# already introduced
```
CHAPTER 1. INTRODUCTION TO PYTHON

# input :
# allows to acquire data from the keyboard
current_year = 2016
age = input("How old are you ? ")
print(age)
type(age)
    #age is a string, so that no maths operation can be performed
    #with this variable
your_age = int(age)
type(your_age)
print('You are born in',current_year - your_age)

Conversion of types :
bool → int,  int → bool,  int → float,  string → int, ...

2.1.2 Conditional instructions
Proceed with file f02CondInstructions.py, in which we consider the following items.

Conditional instructions : if, elif, else
Comparison operators : ==, >, >=, <, <=, !=, is
Predicate : True, False
Boolean operators : and, or, not

a = 5
if a > 0 :  #notice the two points ':'
print("a is greater than 0")

var = 78
if var < 50 :
    print('var is lower than 50')
else :
    print('var is at least 50')

a = 2
b = -3
if a > 0 and b > 0 :
    print("a and b are positive")
elif a > 0 :
    print("only a is positive")

value = input('Enter a value: ')
value = float(value)
if value == 0 :
    print("the value is zero")
elif value != 5 :
    print("the value is not 5")
else :
    print("the value is probably 5")

0 == 0. # comparison of an int and a float --> should return True
7.2 == 5 # --> should return False
7.2 != 5 # 7.2 different from 5 ? --> should return True
w1 = (8 != 9)
w2 = not ( ( ( 8 < 3 ) or ( 7 != 6 ) ) and w1 ) # --> headache...

var = False
if var is True :
    print('var is true')
else :
    print('var is false')

allowed = False
if allowed is not True:
    print("Sorry, you cannot go...")

Exercise 1

A year is called a leap year if it is a multiple of 4, unless it is a multiple of 100. However, it is considered as a leap year if it is a multiple of 400. Precisely:

| If a year is not a multiple of 4, it is not a leap year and we stop there. |
| If it is a multiple of 4, we check if it is a multiple of 100 |
| If it is a multiple of 100, we check if it is a multiple of 400 |
| In that case, it is a leap year |
| Else, it is not a leap year |
| Else, it is a leap year. |

Write a program in Python allowing a user to enter a year and deciding if this year is a leap year or not.

2.1.3 Loops

Proceed with file f03Loops.py

- Loops : while, for
Loops: while, for

```python
x = 0
x_max = 10
sum = 0
print("I am going to count ")
while x <= x_max :
    print(x)
    sum += x
    x = x + 1
print("It is finished")
print("The sum of integers from 0 to",x_max,"is",sum)

mystring = "Hello my dear Axel !"
for letter in mystring:
    print(letter)
```

- range – Notice that by default, indices start from 0 (and not from 1)

```python
for u in range(10) :
    print(u)

for u in range(5,10) :
    print(u)

for u in range(15,10,-1) :
    print(u)

for i in range(10,0,-3):
    print(i)
```

- Break

```python
while 1: # 1 is always true leading to an infinite loop
    letter = input("type 'Q' for leaving : ")
    if letter == "Q":
```
### CHAPTER 1. INTRODUCTION TO PYTHON

- **Continue**

  ```python
  # We compute the sum of integers not divisible by 3 from 1 to 20
  u = 0
  sum = 0
  while u <= 20 :
    if u % 3 == 0 :
      print("We omit the integer", u)
      u += 1
      continue # go back to the while without performing # the remaining lines
    sum += u
  print("with variable u =", u,"the sum is now", sum)
  u += 1
  ```

- **raise an exception**

  ```python
  age = float(input("How old are you ? "))
  print(age)
  if age <= 0 :
    raise Exception('Type_Error : age is not positive')
  elif age > 150 :
    raise Exception('Type_Error : too old , are you sure ?')
  else :
    print("Thank you! Your age is", round(age))
  print("Here is the rest of the program .............")
  print("that will not be executed in case of a 'raise Exception'")
  ```

- **try, except**

  ```python
  while True :
    try :
      x = input("Please enter a number: ") # x is a string
  ```
x = int(x)  # x is converted to integer
print("x=",x)
break  # exit from the loop
except ValueError:  # e.g., '4.3' cannot be converted to int
    print("Oops! that was no valid number. Try again...")
print("Here is the rest of the program ...........")

Exercise 2

Complete the instructions from line 17 to line 19 in file 03Loops.py so as to print the letters of the string ‘mystring’ followed
- either by ‘is a vowel’ if the current letter is a vowel,
- or by ‘is not a vowel’ otherwise.

Exercise 3

Modify the instructions from line 6 to line 14 in file 03Loops.py so as to compute the sum of the squares of integers from x_min = 3 to x_max = 7 in two different ways:
- first by using a ‘while’,
- then, by using a loop ‘for’.

2.2 Functions

A function is a sub-program devoted to some specific tasks. You can give it some parameters (or not) and the function will return some values (or not). E.g., the function can return the inner product of two vectors or can plot the graph of a curve. You must be able to create your own library of tool functions for building software projects. Each function should be precisely commented.

2.2.1 Definition and execution of functions

• Execution from the same file:
  Proceed with file f04FunctionsA.py

```python
# File f04FunctionsA.py
"""""""""""""""""""""""""""""""""""""""""""
A) Definition and use (call) of functions
   in the same file
""""""""""""""""""""""""""""""""""""""""""""""""
""""
"
We first define two functions.
In fact, the second function uses (ie, calls) the first one which is 'visible', because in the same file
"""""""""""""""""""""""""""""""""""""
def product(k1,k2):
  """"k1 and k2 are two non negative integers with 0 <= k1 <= k2
  This function computes the product k1*(k1+1)*...*(k2-1)*k2
  of all integers between k1 and k2
  """
p = 1
for i in range(k1,k2+1):
    p = p * i
return p
""""""""""""""""""""""""""""""""""""""""
```
```python
def Cnk(n, k):
    ""
    n and k are two non negative integers with 0 <= k <= n
    This function computes the combination 'n choose k'
    equal to the ways to choose k elements, disregarding
    their order, from a set of n elements
    ""
    if n-k > k :
        num = product(n-k+1, n)
        den = product(1, k)
        res = num/den
        return res
    else :
        num = product(k+1, n)
        den = product(1, n-k)
        res = num/den
        return res

# File f04FunctionsB1.py
""
B1) Definition of two functions
-------------------------------------------
def product(k1, k2):
    ""
    k1 and k2 are two non negative integers with 0 <= k1 <= k2
    This function computes the product k1*(k1+1)*...*(k2-1)*k2
    of all integers between k1 and k2
    ""
    p = 1
    for i in range(k1, k2+1):
        p = p * i
    return p
def Cnk(n, k):
    ""
    n and k are two non negative integers with 0 <= k <= n
    This function computes the combination 'n choose k'
    equal to the ways to choose k elements, disregarding
    their order, from a set of n elements
    ""
    if n-k > k :
        num = product(n-k+1, n)
        den = product(1, k)
```
res = num/den
return res
else:
    num = product(k+1,n)
    den = product(1,n-k)
    res = num/den
    return res

# File f04FunctionsB2.py

""" ----------------------------------------
B2) Use (call) of functions defined in the file f04FunctionsB1.py)
------------------------------------------- """
from f04FunctionsB1 import Cnk

nMax = 6
for n in range(nMax+1):
    print("\nCoefficients for n =",n,":\") #\n to skip a line
    for i in range(n+1):
        print(Cnk(n,i))

• Execution from the IPython console.

1. In the IPython console → menu Execution → item Configure → Working directory:
   select your current working directory, typically
   "C:\Users\Luke\Documents\Myanmar\NumericalMaths\01IntroPython\PythonScriptsOK"

2. As an example, create (and save it in your current working directory) the Python file “LeapYear.py”
   with the following script

# Acquisition of a year
year = input("Enter a year: ")
year = int(year)
if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    print(year,"is a leap year")
else:
    print(year,"is a not leap year")

3. Finally, execute the following command in the IPython console
   In [11]: runfile('LeapYear.py')

   • from windows system.

2.2.2 Recursive function

A recursive function is a function that calls itself. See the example of the factorial function given in
the following file.
Proceed with file f04FunctionsC.py

# File f04FunctionsC.py

""" Example of a recursive function

...
CHAPTER 1. INTRODUCTION TO PYTHON

# Factorial function: implementation with a loop (while)
def fact(n):
    """ n is a non negative integer
    This function computes the factorial of n
    ""
    res = 1
    while n > 0:
        res = res * n
        n = n - 1
    return res

print(fact(5))
print(fact(0))

# Recursive implementation of the factorial:
# the function factRec() calls itself
def factRec(n):
    """ n is a non negative integer
    This function computes the factorial of n in a recursive way
    ""
    if n <= 1:
        return 1
    else:
        return n * factRec(n-1)

print(factRec(4))
print(factRec(0))

2.2.3 Local and Global variables

Proceed with file f04FunctionsD.py

# File f04FunctionsD.py
"""------------------------------------------------""
Local and global variables
"""------------------------------------------------""
An assignment in a function create a local variable:
"""------------------------------------------------""
def x_equal_42():
    x = 42
    print("inside the function: x =",x)  # displays 42

x = 0
x_equal_42()
print("after the function: x =", x)
    # displays 0: the global variable is not modified

"""------------------------------------------------
Global variables are 'visible' inside the functions:
def display_x():
    print("inside the function: x =", x)

x = 47
display_x()
print("after the function: x =", x)

A global variable can be modified in a function with the command 'global':

def x_equal_42():
    global x # <-- here
    x = 42
    print("inside the function: x =",x) #displays 42

x = 0
x_equal_42()
print("after the function: x =", x) #displays 42

2.2.4 Lambda functions

Proceed with file f04FunctionsE.py

# File f04FunctionsE.py
""
lambda function
""

f = lambda x: x*x
print(f(8))

g = lambda x, y: x + y
print(g(8,9))

# Compare with the equivalent classic functions

def ff (x):
    return x**2
print(ff(8))

def gg (x,y):
    return x+y
print(gg(8,9))

2.2.5 The instruction test: if __name__ == '__main__':

Proceed with file f04nameMain1a.py, f04nameMain1b.py, f04nameMain2.py

# File f04nameMain1a.py
""
Example for the test "if name==main"
""
def function1a():


```python
print("Execution of function 1a")
function1a()
```

```python
# File f04nameMain1b.py

"""
Example for the test "if name==main"
"""

def function1b():
    print("Execution of function 1b")

if __name__ == '__main__':
    function1b()
```

```python
# File f04nameMain2.py

"""
Example for the test "if name==main"
"""

import f04nameMain1a
import f04nameMain1b
print("Execution of file 2")
```

The two files f04nameMain1a.py and f04nameMain1b.py are similar. But the use of the test if __name__ == '__main__': in the second file f04nameMain1b.py avoid the execution of this file when you import it in another file, such as f04nameMain2.py

2.3 Files

Proceed with files f05Files.py

- Change directory

```python
# File f05Files

"""----------------------------------------
Change directory
----------------------------------------"""

"""
Remark : we can use directly some windows commands, such as:

```
pwd  --> print working directory
ls   --> list directory contents
```

Directory paths :
Windows uses backslashes (\) to separate directories in file paths:
'C:\Users\Luke\Documents\Luc\Myanmar\NumericalMaths'
whereas Python uses two backslash (\\)
'C:\\Users\\Luke\\Documents\\Luc\\Myanmar\\NumericalMaths'
Here, we can use two backslash (\\) or one slash (/)

"""

import os  # import operating system

os.getcwd()  # print the current working directory
    # with two backslash (\\): similar to pwd
os.chdir("C:\\Users\\Luke\\Documents\\Luc")  # change the directory
```
os.getcwd()

# --> now, we can use one slash to change the directory:
myPath1 = "C:/Users/Luke/Documents/Luc/Myanmar"
myWorkingPath = myPath1 +
                  "/NumericalMaths/01IntroPython/PythonScriptsOK"
os.chdir(myWorkingPath)

# --> relative path:
os.chdir('../../02MachineNumbers') # 2 parent directories, 1 child
os.getcwd()

• Opening and closure of an existing file

"""----------------------------------------
Opening and closure of an existing file
----------------------------------------"""
os.chdir(myWorkingPath) # contains our testing files
my_file = open("file0.txt", "r") # opening in mode "r" (read)
type(my_file)
content = my_file.read() # read the content 'my_file'
type(content)
print(content)
my_file.close() # close the file

• Modifying (writing in) or Creating an existing file

"""----------------------------------------
Modifying (writing in) an existing file
Creating a file
- First, we need to open the existing file
  mode "w" overwrites the possible content of the file
  mode "a" (append) adds what is written at the end of the file
  (\n to skip a line)
- If the file doesn’t exist, it will be created
----------------------------------------"""
my_file = open("file1.txt", "w") # Oops, I overwrite everything!
my_file.write("writing in a file via Python !!!")
    # return the number of characters
my_file.close()

my_file = open("file1.txt", "w") # again, we overwrite everything
string = "I am going to write in a file... "
my_file.write(string)
my_file.close()

my_file = open("file1.txt", "a") # to write at the end of the file
my_file.write("\nDoes it work ?\n--> YES, IT WORKS !!")
my_file.close()
# Repeat the last 3 instructions several times
# and then check the text file
2.4 Overview on Modules and Libraries

Usual constants and mathematic functions are not known in basic Python. E.g., try the following instructions.

```
Traceback (most recent call last):

File "<ipython-input-21-68f7b1e53523>", line 1, in <module>
  pi
NameError: name 'pi' is not defined

In [12]: sqrt(2)
Traceback (most recent call last):

File "<ipython-input-22-40e415486bd6>", line 1, in <module>
  sqrt(2)
NameError: name 'sqrt' is not defined
```

Most of mathematical functions, usual constant, graphical tools, numerical methods,... have to be imported from libraries or modules. We give below some examples of the use of these modules. Nevertheless, a more detailed overview of the modules `numpy` and `matplotlib` is given in section 3 and section 4.

Proceed with file `f06LibrariesA.py`

- The module `maths` : Maths library

```
"""
math (mathematic library) """
import math as math
v = math.pi # the constant 'pi' is reached from the library 'math'
print(v)

u = math.sqrt(2) # the function sqrt() is reached from the
# library 'math'
print(u)

uu = u**2
print(uu) # not exactly equal to 2 (...?)

# another way to proceed is to import only desired "tools":
from math import pi
print("pi =",pi)

from math import sin, sqrt, pi
x = sin(pi/4)
print(x,"=",sqrt(2)/2)
```

- The module `numpy` : Numerical Python

NumPy is the fundamental package for scientific computing in Python. NumPy’s main object is the homogeneous multidimensional array.

https://docs.scipy.org/doc/numpy-dev/contents.html
CHAPTER 1. INTRODUCTION TO PYTHON

```python
import numpy as np

u = np.array([2, 3, 5])
v = u+2
w = u**2
y = np.cos(u)  # most of math functions are also defined in numpy
type(u)
help(type(u))

# np.arange(start, stop, step)
# spaced values by step <step> within the half–open interval
#   [start, stop)
x = np.arange(0, 2, 0.4)
print(x)
y = np.arange(3, 9, 2)
z = np.arange(5, 3, -0.2)
type(z)

# np.linspace(start, stop, N)
# N evenly spaced values over the interval [start, stop]
t = np.linspace(-1, 2, 10)  # Return 10 evenly spaced numbers
    # in [-1, 2]
print(t)
type(t)
y = np.sin(t)
```

- The module scipy: Scientific Python
  SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python.
  http://docs.scipy.org/doc/scipy/reference/tutorial/

```python
from scipy.interpolate import interp1d
import numpy as np
import matplotlib.pyplot as plt

help(interp1d)

# plotting of the function to be interpolated:
t = np.linspace(0, 10, num=100, endpoint=True)
yt = np.cos(-t**2 / 9.0)
# sampling of the function:
x = np.linspace(0, 10, num=11, endpoint=True)  # 11 sampled values
y = np.cos(-x**2 / 9.0)
# interpolation:
```
f1 = interp1d(x, y)  # linear interpolation
f3 = interp1d(x, y, kind='cubic')  # cubic interpolation
# plotting of the two interpolants
plt.plot(t, y, 'ro', x, y, 'ro', t, f1(t), '-', t, f3(t), '--
plt.legend([''initial function'', ''data'', ''linear'', ''cubic''],
loc='best')

- The modules `matplotlib` and `matplotlib.pyplot`: Graphical libraries
  Matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB
  http://matplotlib.org/users/pyplot_tutorial.html

```python
import numpy as np
import matplotlib.pyplot as plt # for graphic purposes
plt.xlabel('x-axis: abscissas')
plt.ylabel('y-axis: ordinates')
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()

plt.clf()  # clear the current figure
x = np.arange(0, 2 * np.pi, 0.1)
y = np.sin(x)
plt.plot(x, y)
x = np.arange(0, 2 * np.pi, 0.5)
y = np.sin(x)
plt.plot(x, y, 'go')
```

- The module `sympy`: Symbolic Python
  Symbolic computation deals with the computation of mathematical objects symbolically. This means that the mathematical objects are represented exactly.
  http://docs.sympy.org/latest/tutorial/

```python
import numpy as np
x = np.sqrt(8)  # result = 2.8284271247461903
```
import sympy as sympy
y = sympy.sqrt(8)  #result = 2*sqrt(2)

from sympy import symbols
x, y = symbols('x y')
expr = x + 2*y
expr2 = expr + 1 - x

from sympy import expand, factor
expr3 = x*expr
expanded_expr = expand(expr3)
factor(expanded_expr + x)

from sympy import diff, sin, exp
x = symbols('x')
y = diff(sin(x), x)
z = diff(sin(x)*exp(x), x)

• The module random

    import random as rd
    import numpy as np

    N = 10
    sum = 0
    for i in range(N):
        x = rd.random()  #return floating value in the interval [0, 1)
        print(x)
        sum += x
    average = sum / N
    print("average =", average)
    v = rd.randint(6)  # random integer chosen from range(6)
    print(v)
    rd.choice(['apple', 'pear', 'banana'])
    rd.sample(range(100), 10)  # Chooses 10 unique random elements in range(100)

    # random values can also be generated with the numpy module
    # 1) random samples from a uniform distribution over [0,1) :
    np.random.rand(4)  # random values in an numpy.ndarray
    np.random.rand(3,2)
    # 2) normal (Gaussian) distribution :
    mu, sigma = 0, 0.1  # mean and standard deviation
    s = np.random.normal(mu, sigma, 10)

• The module time

    import time

    start = time.time()
    # do some calculations
    time.sleep(2)
    end = time.time()
    print("execution time =", end-start)
import time

print("Hello World, please wait 2 seconds...")
time.sleep(2)
print("Thank you")

start = time.time()  # number of seconds from
                    # January 1st 1970, 00:00:00.
print("start =", start)
time.sleep(2)  # we sleep for 2 seconds...
end = time.time()
print("end =", end)
print("Duration =", end-start)

• Modules system : os, sys, ...

2.5 Some structured types in Python

2.5.1 String

The class str (string = string of characters) is a class in the module builtins. The module builtins provides direct access to all ‘built-in’ identifiers of Python. The class str possesses its own methods which can be viewed as specific functions with a specific syntax as stated below.

A function usually works as follows

\[
\text{output} = \text{fct(st1)}
\]

where fct is a function with input parameter the string st1.

A string method works as follows

\[
\text{output} = \text{st1.fct()}
\]

where fct() is a string method, that applies only to a string.

Proceed with file f10String.py, in which we consider the following items.

• Introduction to the class string
• Concatenation of strings
• Access to (and modification of) string elements
• Example of a string method : lower and upper
• Another string method : the method format
• The string methods : split and join

These two methods involve the type list and thus will be presented in the next section.

```python
# File f10String

"""""""""""""""""""""""""""""""""

The class string

"""""""""""""""""""""""""""""""""""

str1 = 'first example of a string...'
print(str1)
type(str1)
len(str1)  # length of the string
help(str)

""""""""""""""""""""""""""""""""""

Concatenation of strings
```
CHAPTER 1. INTRODUCTION TO PYTHON

---

```python
str1 = 'The weather'
str2 = "is fine today."
str3 = str1 + str2
print(str3)

---

Access to (and modification of) string elements

```python
str1 = "he will come asap"
str1[0] #the first element of the string
str1[2:] #elements of the string starting from the third one
str1[0] = "H" #doesnot work: a string object does not support item assignment
str2 = "H" + str1[1:]
print(str2)
str3 = "They" + str1[2:]
print(str3)
```

#example
mystring = "Hello guys"
i = 0
while i < len(mystring):
    print(mystring[i])
i += 1

---

Example of a string method : lower and upper

```python
str1 = 'Text in UPPERCasE'
str2 = str1.lower() #the string is converted in lowercase
str3 = str1.upper() #the string is converted in uppercase
print(str1, str2, str3)
help(str.lower)
```

#example
mystring = str() #an empty string
while mystring.lower() != "q":
    print("Type 'q' or 'Q' to leave...")
    mystring = input()
print("thank you !")

---

Another string method : the method format

```python
firstname = "Johnny"
name = "Hallyday"
age = 73
st0 = "My name is {0} {1} and I am {2} years old"
st1 = st0.format(firstname, name, age)
print(st1)
print("My name is {0} {1} ({3} {0} for the administration)" \
    " and I am {2} years old"
    .format(firstname, name, age, name.upper()))
```

---

The string methods : split and join
these two methods involve the type list
and thus will be presented in the next section
```
2.5.2 List

String methods does not alter the original object: they all return a new object which is the modified string. For a list, it is the opposite: the methods of lists modify the original object and does not return a new object.

Proceed with file `f11List.py`, in which we consider the following items.

- Creation of a List: Square brackets [] define lists
- Access to (and modification of) elements of a list
- The method append
- The method insert
- Concatenation of lists: the method extend
- Remove elements of a list: del, and the methods remove & pop
- Go through a list with the function enumerate
- Conversion string → list with the string method split
- Conversion list → string

```python
# File f11List
"""
As a reminder:
    Square brackets [] define lists
    Parenthesis ( ) define tuples
    Curly brackets {} define dictionaries
"""

""
Creation of a List
""
my_list = list() # create an empty list
type(my_list)
len(my_list)
my_list
print(my_list)

# or:
my_list = [] # create an empty list

# simple example:
my_list = [1, 2, 3, 4, 5] # a list with 5 objects
print(my_list)

# another example:
my_list1 = ['c', 'f', 'm']
my_list2 = [-7, 3.5, "a string", [], my_list1]
len(my_list1)
type(my_list2)
len(my_list2)

"""
Access to (and modification of) elements of a list
"""
```
my_list2[4]  # the fourth element of the list
my_list2
my_list1[1] = 'new element'  # the first element is replaced by another one
my_list1
my_list2

The method append

my_list = [1, 2, 3]
my_list.append(-56)  # we add 56 at the end of the list
my_list  # becomes: [1, 2, 3, -56]

REMARK:
String methods does not alter the original object: they all return a new object which is the modified string. For a list, it is the opposite: the methods of lists modify the original object and does not return a new object.

string1 = "A small sentence"
string2 = string1.upper()  # we uppercase string1
string1
string2

list1 = [1, 5.5, 18]
list2 = list1.append(-15)  # we add -15 at list1
list1  # display of list1
list2  # nothing?
print(list2)  # None: list2 does not exist

The method insert

my_list = ['a', 'b', 'd', 'e']
my_list.insert(2, 'c')  # we insert 'c' at position with index 2
print(my_list)
my_list.insert(2, 'c')
print(my_list)

Concatenation of lists: the method extend

my_list1 = [3, 4, 'r']
my_list2 = [8, 'text', 10]
my_list1.extend(my_list2)  # we insert my_list2 at the end of my_list1
print(my_list1)  # my_list1 is the concatenation of the 2 lists
print(my_list2)  # my_list2 is not modified

my_list1 = [3, 4, 'r']
my_list2 = [8, 'text', 10]
newlist = my_list1 + my_list2
print(my_list1)  # my_list1 is not modified
print(my_list2)  # my_list2 is not modified
print(newlist)  # newlist is the concatenation of the 2 lists

my_list1 += my_list2  # is identical to extend

"""
Remove elements of a list: del, remove, pop

```
#del
var = 34
var
del var
var #==> NameError: name 'var' is not defined

my_list = [-5, -2, 1, 4, 7, 10]
del my_list[0]  #deletes the first element of the list
my_list
del my_list[2]  #deletes the third element of the list
my_list

#the method remove(value): deletes only the first occurrence of value
my_list = [31, 32, 33, 32, 35, 32]
my_list.remove(32)
my_list

#the method pop() removes (and returns) last object from the list
aList = [123, 'xyz', 'zara', 'abc', 2016]
aList.pop()  #removes last object of the list
aList.pop(2)  #removes object with index 2 of the list
```

Go through a list with the function 'enumerate'

```
my_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
for elt in my_list:
    print(elt)

#with enumerate:
for elt in enumerate(my_list):
    print(elt)  #each elt is a couple (position in my_list, associated value)
    #we will see later that this couple is in fact a tuple

for i, elt in enumerate(my_list):
    print("At position {} is {}.").format(i, elt))

list2 = [
    [1, 'a'],
    [4, 'd'],
    [7, 'g'],
    [26, 'z']
]
for elt in enumerate(list2):
    print(elt)
for nb, letter in list2:
    print("Letter {} is the {}e of the alphabet".format(letter, nb))
```

Conversion string --> list with the string method 'split'

```
string1 = "Hello ladies and gentlemen"
    #two blank spaces between 'ladies' and 'and'
c1 = string1.split(" ")
c2 = string1.split()  #default
c3 = string1.split('a')
string1
type(c1)
```
print(c1)
print(c2)
print(c3)
help(str.split)

"""
Conversion list —> string
"""
list0 = ['May', 'I', 'help', 'you']
str3 = " ".join(list0)  # strange syntax...
help(str.join)

Exercise 4
Write a function which takes in input parameter a floating number ‘fl’ and return a string representing the truncation of this floating number ‘fl’ with a maximum of three digits for its decimal part, and which replaces the dot ‘.’ by a comma ‘,’
Example : floatdisplay(51.589106) should return 51,589

2.5.3 Tuple
Tuples are lists that can not be modified: once created, it is not possible to add or to remove an element from a tuple. Notice that to create a tuple with a unique element, it is necessary to type a comma after this unique element.

Proceed with file f12Tuple.py, in which we consider the following items.

- Creation of a tuple: Parenthesis ( ) define tuples
- Utilization: multiple assignments
- Utilization: a function returning several values
- The methods count and index

# File f12Tuple
"""
As a reminder :
Square brackets [ ] define lists
Parenthesis ( ) define tuples
Curly brackets { } define dictionaries
"""

"""
REMARK.
Tuples are lists that can not be modified: once created, it is not possible to add or to remove an element from a tuple
Remark: to create a tuple with a unique element, it is necessary to type a comma after this unique element
"""

"""
Creation of a tuple
"""
empty_tuple = ()
nonempty_tuple = (1, )  # equivalent to the following line
nonempty_tuple = 1,
my_tuple = (1, 2, 5)
type (my_tuple)
help (tuple)

""
Utilization: multiple assignments
""
a, b = 3, 4
a b
(a, b) = (3, 4) #equivalent to: a, b = 3, 4

""
Utilization: a function returning several values
""
def EuclideanDivision(divident, divisor):
    """The Euclidean division is the process of division of two integers
    (division of the integer 'divident' by the integer 'divisor'),
    which produces a 'quotient' and a 'remainder' with the property that
    0 <= remainder < abs(divisor).
    Precisely:
    q, r = EuclideanDivision(a, b)
    such that: a = b q + r and 0 <= r < |b|
    ""
    quotient = divident // divisor
    remainder = divident % divisor
    return quotient, remainder

#Examples:
quotation, remainder = EuclideanDivision(43, 5)
quotation
remainder
back = EuclideanDivision(37, 8)
type(back)

""
REMARK about the interest of commenting your personal functions (and programs)
""
help(EuclideanDivision) #provides your own comments!

""
The methods count and index
""
u = (-1, 2, "a", "r", 6, 2, 'r', 7, -1, "r", 2, "r")
u.count(2) #return the number of occurrences of 2
u.count('r') #return the number of occurrences of 'r'
u.index(2) #return first index value of 2
u.index('r') #return first index value of 'r'

""
Miscellaneous: list of tuples and zip
""
a = [2, 3, 4] #first list
b = [5, 6, 7] #second list
w = list(zip(a, b)) #create a list of tuples
    # w = [(2, 5), (3, 6), (4, 7)]
w[0] # (2, 5)
w[2][1] # 7
2.5.4 Dictionary

A dictionary is a catalog (or a list) of items that can be reached by keys, and not necessarily by indices as in a list.

Proceed with file f13Dict.py, in which we consider the following items.

- Creation of a tuple: Curly brackets \{\} define dictionaries
- Creation of a dictionary by keys (the key is between square brackets)
  Access and modification
- Direct assignment of a dictionary
- Remove keys of a dictionary:
  with del
  with the method pop

```python
# File f13Dict

"""
As a reminder:
  Square brackets [ ] define lists
  Parenthesis ( ) define tuples
  Curly brackets { } define dictionaries  
"""

"""
REM A R K.
A dictionary is a catalog (or a list) of items that can be reached by keys, and not necessarily by indices as in a list  
"""

"""
Creation of a dictionary:
"""
my_dictionary = dict()  # an empty dictionary
my_dictionary = {}  # equivalent to the previous line
my_dictionary
help(dict)

"""
Creation of a dictionary by keys:
  (the key is between square brackets)
Access and modification
"""
my_dict = {}  # first, create an empty dictionary
my_dict["pseudo"] = "Gandalf"  # pseudo is the key of the element Gandalf
my_dict["password"] = "easypwd"  # password is the key of the element easypwd
my_dict  # display of the dictionary
my_dict["pseudo"]  # display of the element associated with the key pseudo
my_dict["password"]  # display of the element associated with the key password
my_dict["pseudo"] = "Olorin"  # modification of the element associated with pseudo
my_dict
```
### Another exemple

```python
newdict = {}
newdict[0] = "a"
newdict[1] = "e"
newdict[2] = "i"
newdict[3] = "o"
newdict[4] = "u"
newdict[5] = "y"
newdict  # dictionary of vowels whose keys are the indices from 0 to 5
# similar to the following list
vowels = ['a', 'e', 'i', 'o', 'u', 'y']
```

### New exemple : here the keys are tuples

```python
chessboard = {}
chessboard[('a', 1)] = "white rook"
chessboard[('b', 1)] = "white knight"
chessboard[('c', 1)] = "white bishop"
chessboard[('d', 1)] = "white queen"
chessboard[('e', 1)] = "white king"
chessboard[('f', 1)] = "white bishop"
chessboard[('g', 1)] = "white knight"
chessboard[('h', 1)] = "white rook"
chessboard
```

#### Direct assignment of a dictionary:

```python
""
""
cupboard1 = {
"shirts": 8, 
"pants": 6, 
"socks": 7
}
#is equivalent to :
cupboard2 = {}
cupboard2["shirts"] = 8
cupboard2["pants"] = 6
cupboard2["socks"] = 7
""
""
```

#### Remove keys of a dictionary:

```python
""
with del
with the method pop
""
""
#with del :
cupboard1 = {"shirts": 8, "pants": 6, "socks": 7}
del cupboard1["shirts"]
cupboard1

#with the method pop which removes the key and returns the removed element
cupboard1 = {"shirts": 8, "pants": 6, "socks": 7}
cupboard1.pop("chemise")
cupboard1
```

### As a reminder:
- Square brackets [] define lists
- Parenthesis ( ) define tuples
- Curly brackets { } define dictionaries
2.6 Class

https://docs.python.org/2/tutorial/classes.html

Proceed with files f20Class.py, f21Class.py, f22Class.py, f23Class.py, f24Class.py, in which we consider the following items.

- Class definition syntax.
  Basically a class is a Python structure with
  - attributes
  - methods, i.e., functions that work directly with the objects of the class
  An object is an instance of a class.
  Remark: most of the Python objects we have already encountered are classes instances, e.g.,
  type: help(list), help(tuple), help(dict), help(float), help(str),...

- Class variables, Instance variables and Constructor methods.
  - A class variable is shared by all instances of the class
  - An instance variable is unique to each instance
  - A constructor method allows to define instance variables

- Instantiation and modification of attributes.
  The instantiation of an object of a class requires the instantiation of each instance variable
  (attribute) specified by the constructor method __init__(self, ...)

- Inheritance

- About the parameter self:
  - Attributes are contained in the objects (i.e., instances of the class).
  - Methods are contained in the class that defined the object.

```python
# File f20Class
# https://docs.python.org/2/tutorial/classes.html

""
Class definition syntax :
- Basically a class is a Python structure with
  -> attributes
  -> methods, i.e., functions that work directly with the objects of the class
- An object is an instance of a class
""

class CL1(object):
  ""
  An example of class with two attributes and two methods
  ""
  comment = "my first example !" #attribute
  nb = 0 #another attribute
  def f1(self):
      #first method
      return 'hello world'
  def f2(self):
      #second method
      self.nb += 1
      return self.nb

  t = CL1() #t is an instance of the class CL1 (t is an object)
  t.comment
  t.nb
  t.__doc__
  t.f1() #display 'hello world'
  t.f2() #return 1
  t.f2() #return 2
```
CHAPTER 1. INTRODUCTION TO PYTHON

t.f2()  #return 3

t1 = CL1()  #another instance of the class CL1
t1.f2()  #return 1

"""__________________________.
REMARK:
most of the Python objects we have already encountered are classes :
"""

help(list)
help(tuple)
help(dict)
help(float)
help(str)

# File f21Class
# https://docs.python.org/2/tutorial/classes.html

"""__________________________
Class variables, Instance variables and Constructor methods :
− a class variable is shared by all instances of the class
− an instance variable is unique to each instance
− a constructor method allows to define instance variables
"""
class Dog(object):
    kind = 'canine'  # class variable shared by all instances
    def __init__(self, name):  # constructor method
        self.name = name  # instance variable unique to each instance

d = Dog('Fido')
e = Dog('Buddy')
d.kind  # shared by all dogs
e.kind  # shared by all dogs
d.name  # unique to d
e.name  # unique to e

# Mistaken use of a class variable :
class Dog(object):
    tricks = []  # class variable shared by all instances
    def __init__(self, name):  # constructor method
        self.name = name
    def add_trick(self, this_trick):
        self.tricks.append(this_trick)

d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
d.tricks  # ['roll over', 'play dead']
e.tricks  # ['roll over', 'play dead']
d.name  # 'Fido'
e.name  # 'Buddy'

# It is better to use an instance variable :
class Dog(object):
    def __init__(self, name):
        self.name = name
        self.tricks = []  # creates a new empty list for each dog
    def add_trick(self, this_trick):
self.tricks.append(this_trick)

d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
d.tricks # ['roll over']
e.tricks # ['play dead']

# File f22Class
# https://docs.python.org/2/tutorial/classes.html
"""
Instantiation and modification of attributes :
"""

class Person(object):
    def __init__(self):
        """ Constructor method of our class.
        Each attribute is instantiated by a default value
        """
        self.name = "Dupont"
        self.firstname = "Jean"
        self.age = 33
        self.address = "Toulouse"

Id = Person()
Id.name = "Ravel" #attributes can be instantiated and modified
Id.firstname = "Maurice"
Id.age = 54
Id.address = "Paris"
Id.address = "unknown"
Id.address

"""
In this example, the instantiation of an object of the class Person
requires the instantiation of instance variables (attributes)
'name', 'firstname', and 'birth_date'
specified by the constructor method __init__(self,...)
"""

class Person(object):
    def __init__(self, name, firstname, birth_date):
        """ Constructor method of our class.
        """
        self.name = name
        self.firstname = firstname
        self.birth_date = birth_date
        self.address = "Toulouse" #value by default
        self.age = 0 #value by default

    def update_age(self, year):
        self.age = year - self.birth_date

Id = Person() #produces an error
Id1 = Person('Ravel', 'Maurice', 1875)
Id1.name
Id1.address
Id1.address = "Paris"
Id1.address
Id1.update_age(2016)
Id1.age
Id2 = Person('Berlioz','Hector',1803)
Id2.name
Id2.address
Id2.address = "France"
Id2.address

# File f23Class
# https://docs.python.org/2/tutorial/classes.html
""
Inheritance:
""
class Dog(object):
    def __init__(self, name):
        self.name = name
        self.tricks = [] # creates a new empty list for each dog
    def add_trick(self, this_trick):
        self.tricks.append(this_trick)
class RacingDog(Dog):
    def __init__(self, name, successes):
        self.name = name
        self.tricks = []
        self.successes = successes
    def update_successes(self, successes_thisyear):
        self.successes += successes_thisyear
u = RacingDog("Medor", 11)
help(u)
u.name
u.tricks
u.successes
u.add_trick('roll over') # a method of the class Dog()
u.tricks
u.update_successes(5) # a method of the class RacingDog()
u.successes

# File f24Class
# https://docs.python.org/2/tutorial/classes.html
""
About the parameter self:
- Attributes are contained in the objects (i.e., instances of the class).
- Methods are contained in the class that defined the object.
    Thus (see below):
    tab.write("text")
is identical to
    MyNoteBook.write(tab, "text")
""
class MyNoteBook: # by default : MyNoteBook(object)
    """Class defining a page on which we can write messages, read the written messages and also erase all messages, by means of methods.
    The modified attribute is page.
    """
    def __init__(self):
        """Initially the page is empty""
        self.page = ""
    def write(self, message):
        """This method allows to write on the attribute page"""
3 Linear algebra with numpy

3.1 Vectors and matrices

```python
import numpy as np

# Vectors and matrices with numpy.array
# = one-dimensional or two-dimensional arrays

# a vector:
v = np.array([-4, 17, 0, 11])
v
print(v)

# a matrix:
m = np.array([[1, 2], [3, 4], [5, 6]])
m
print(m)
```

CHAPTER 1. INTRODUCTION TO PYTHON

2) Functions type, np.size, np.shape

```python
# compare with:
type([-4,17,0,11])

type(m)

sizev = np.size(v)  # 4
sizem = np.size(m)  # 6

shapev = np.shape(v)  # (4,)
shapem = np.shape(m)  # (3, 2)
```

3) Some particular matrices:

```python
np.zeros, np.ones, np.eye, np.diag
np.arange, np.linspace
```

```python
z1 = np.zeros(5)
z2 = np.zeros((2,3))
o1 = np.ones(6)
o2 = np.ones((3,4))
e1 = np.eye(4)

m = np.array([[1, 2, 3],[4,5,6],[7,8,9]])  # a square matrix
d = np.diag(m)  # the main diagonal
np.diag(m, 1)  # the first upper diagonal
np.diag(m, -1)  # the first lower diagonal
A = np.diag(d)  # a matrix with main diagonal = d
B = np.diag(d,1)  # a matrix with first upper diagonal = d
C = np.diag(d,-2)  # a matrix with second lower diagonal = d
```

```python
a = np.arange(5)
# array([0, 1, 2, 3, 4])
b = np.arange(2,8)
# array([2, 3, 4, 5, 6, 7])
c = np.arange(5,2,-0.5)
# array([ 5.,  4.5,  4. ,  3.5,  3. ,  2.5])
t = np.linspace(-2,3,10)
# 10 values uniformly distributed between -2 and 3
```

4) Access and slicing:

```python
v = np.array([-4, 17, 0, 11, -4, 8])
u = v[3]
w = v[1:4]
```
CHAPTER 1. INTRODUCTION TO PYTHON

\[ M = \text{np.array([[4, 1, 2, -4],[8, 3, 4, -5],[5, 6, -4, 2]])} \]
\[ e = M[1,2] \]
\[ N = M[1:3,1:3] \]

# warning :
\[ U = M[:,2] \] # column 2 (but written as a one-dimensional array
# ==> horizontally)
\[ V = M[0,:] \] # line 0

3.2 Concatenation of matrices

"""----------------------------------------
5) Concatenation of matrices:
----------------------------------------"""
import numpy as np
A = np.array([[4, 1, 2, -4],[8, 3, 4, -5]])
B = np.array([[-1, -1],[-1, -1]])
C1 = np.array([4,4,4,4])
C2 = np.array([4,4,4,4])
D = np.array([[5,5,5,5],[6,6,6,6],[7,7,7,7]])
np.shape(A) # (2, 4)
np.shape(B) # (2, 2)
np.shape(C1) # (4,)
np.shape(C2) # (1, 4)
np.shape(D) # (3, 4)

# concatenation along axis=0 ("vertically")
# or axis=1 ("horizontally")
# the input arrays dimensions must agree along the axis
E = np.concatenate((A,C1), axis=0) # wrong
F = np.concatenate((A,C2,D), axis=0) # OK
G = np.concatenate((A,C2,D), axis=1) # wrong
H = np.concatenate((A,B,A), axis=1) # OK

# concatenation of vectors:
# C1 is viewed as a vector
C = np.append(C1,2*C1)
# array([4, 4, 4, 4, 8, 8, 8, 8])

3.3 Operations on matrices

"""----------------------------------------
6) Basic operations:
    sum,
    product by a scalar,
    term by term product
----------------------------------------"""
import numpy as np
v1 = np.array([-4,17,0,11])
v2 = np.array([7,-14,3,-8])
w1 = 5 * v1
w2 = v1 + v2
w3 = v1 * v2

M1 = np.array([[1, 2],[3,4],[5,6]])
M2 = np.array([[5, 0],[1,2],[-1,3]])
U1 = -2 * M1
U2 = M1 + M2
U3 = M1 * M2

7) Matricial product:
    with numpy.dot()
----------------------------------------
A = np.array([[4, 1], [2, -3], [-1, 3], [4, -2]])
B = np.array([[2, 1], [3, -1]])
C = np.dot(A,B)
D = np.dot(B,A)  # produces an error

8) Transposition of a matrix:
----------------------------------------
A = np.array([[4, 1], [2, -3], [-1, 3], [4, -2]])
AT = A.T

3.4 Complex matrices

9) Complex matrix and conjugate:
----------------------------------------
B1 = np.array([[2-1j, 1+2j], [3-2j, -1]])
B2 = np.conj(B1)
B = B1 + B2
C = np.conj(B1).T

3.5 Copy and Hard copy

Remarks about copy and Hard copy:
----------------------------------------

import numpy as np
A = np.array([1,2,3,4])
A  # array([1, 2, 3, 4])

# Copy (in fact, this is just a link copy):
B = A  # create a new name B to the existing object already named A
B  # array([1, 2, 3, 4])
A  # array([ 1, 2, -7, 4])
3.6 Display of an array

Display (print) of an array:

```python
import numpy as np
u = np.linspace(1, 20, 20)
print(u)
# [1. 2. 3. ..., 18. 19. 20.]
np.set_printoptions(threshold=np.nan)  # => print all values of an array
print(u)
# [1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20.]
```

3.7 Save an array to a text file

Save (and then load) an array to a text file:

```python
import os  # import operating system
myPath = "C:/Users/Luke/Documents/Luc/Myanmar/NumericalMaths"
myWorkingPath = myPath + "/01IntroPython/PythonScriptsOK"
os.chdir(myWorkingPath)

import numpy as np
x = np.array([-1.5, 4., -2.5])
y = x ** 2
# Creation of a data file "data.txt"
# and writing of the array x and y in this text file:
U = (x, y)
np.savetxt('data.txt', U, fmt='%1.8e')
# Reading of the array x and y in the file "data.txt"
(x, y) = np.loadtxt('data.txt')
```

3.8 Determinant and Inverse

10) Determinant:

```python
with numpy.linalg.det()
```

```python
import numpy as np
A = np.array([[4, 5], [2, 3]])
dt = np.linalg.det(A)
```
CHAPTER 1. INTRODUCTION TO PYTHON

# or:
from numpy.linalg import det
dt = det(A)

""
11) Inverse:
   with numpy.linalg.inv()
   ""
A = np.array([[4, 5], [2, 3]])
A1 = np.linalg.inv(A)
# or:
from numpy.linalg import inv
A1 = inv(A)
# then check:
np.dot(A, A1)

3.9 Resolution of a linear system

""
12) Resolution of a linear system:
   with numpy.linalg.solve()
   ""
import numpy as np
A = np.array([[4, 5], [2, 3]])
b = np.array([14, 8])
x = np.linalg.solve(A, b)
# or:
from numpy.linalg import solve
x = solve(A, b)

3.10 Eigenvalue

""
13) Eigenvalue:
   with numpy.linalg.eig()
   ""
import numpy as np
A = np.array([[1, 1, -2], [-1, 2, 1], [0, 1, -1]])
D, V = np.linalg.eig(A)
# or:
from numpy.linalg import eig
D, V = eig(A)
# eigenvalues of the matrix A are in D
# associate eigen vectors are the columns of V
D
Out[1]: array([ 2., 1., -1.])
V
Out[2]:
array([[ 3.01511345e-01, -8.01783726e-01, 7.07106781e-01],
       [ 9.04534034e-01, -5.34522484e-01, -1.92296269e-16],
       [ 3.01511345e-01, -2.67261242e-01, 7.07106781e-01]])

52
3.11 Shape, Nan, Inf

```python
import numpy as np

U = np.arange(1, 25)
U

U.shape

U.shape = (3, 8)  # we modify the shape of U
U

a = np.arange(3, dtype=float)
a[0] = np.nan
a[1] = np.inf
a[2] = -np.inf
a

np.isnan(a[0])  # True
np.isinf(a[1])  # True
np.isinf(a[2])  # True
```

3.12 The class matrix of numpy

```python
import numpy as np

# Data = array_like
# np.matrix is similar as np.array
A1 = np.array([[4, 1, 2, -4], [8, 3, 4, -5]])
A2 = np.matrix([[4, 1, 2, -4], [8, 3, 4, -5]])

type(A1)  # numpy.ndarray

np.shape(A1)  # (2, 4)

# Data = string
# commas or spaces separate columns, semicolons separate rows
A = np.matrix('1 2; 3 4')
B = np.matrix('5.0 6.0')
C = np.matrix('-1 ; -2 ; -3')
D = np.matrix('1.0 2.0; 3.0 4.0; 5.0 6.0')
E = np.matrix('-1 -2; 1 2; 3 -2')
type(A)  # numpy.matrixlib.defmatrix.matrix
np.shape(A) # (2, 2)

# sum and product of matrices
2*D - 3*E
A*B  # dimensions do not agree
B*A  # OK
D*A  # OK

# power, determinant and inverse of a square matrix
A**3
from numpy.linalg import det
dt = det(A)
from numpy.linalg import inv
U = inv(A)
# or
U = A**(-1)

# Concatenation
np.concatenate((A, B))
np.concatenate((C, D), axis=1)
# or with vstack and hstack :
np.vstack((A,B))  # stack them vertically
np.hstack((C,D,C,D))  # stack them horizontally

3.13 Exercises
Exercise 5
Write in the simplest way the following vectors and matrices.

\[
v1 = (5 \ 6 \ 7 \ 8 \ 9 \ 10)
v2 = (0 \ 0 \ 0 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 1 \ 1 \ 1 \ 1)
v3 = (0 \ 1 \ 2 \ 3 \ 4 \ 9 \ 7 \ 5 \ 3 \ 1)
\]

\[
M1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}
M2 = \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 8 & 6 & 4 & 2 & 0 \\ 8 & 6 & 4 & 2 & 0 \end{pmatrix}
M3 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}
\]
Exercise 6

Write in the simplest way the following matrix.

\[
M_4 = \begin{pmatrix}
2. & 1. & 0. & 0. & 0. & 0. & 0. \\
1. & 4. & 1. & 0. & 0. & 0. & 0. \\
0. & 1. & 4. & 1. & 0. & 0. & 0. \\
0. & 0. & 1. & 4. & 1. & 0. & 0. \\
0. & 0. & 0. & 1. & 4. & 1. & 0. \\
0. & 0. & 0. & 0. & 1. & 4. & 1. \\
0. & 0. & 0. & 0. & 0. & 1. & 2.
\end{pmatrix}
\]

Exercise 7

Triangle of Pascal. Write a program in Python computing each line “\( n \)” of the triangle of Pascal for \( 0 \leq n \leq N_{\text{max}} \) as below.

Enter \( N_{\text{max}} : 7 \)

\[
\begin{align*}
n = 0 &: \ [ 1. ] \\
n = 1 &: \ [ 1. \ 1. ] \\
n = 2 &: \ [ 1. \ 2. \ 1. ] \\
n = 3 &: \ [ 1. \ 3. \ 3. \ 1. ] \\
n = 4 &: \ [ 1. \ 4. \ 6. \ 4. \ 1. ] \\
n = 5 &: \ [ 1. \ 5. \ 10. \ 10. \ 5. \ 1. ] \\
n = 6 &: \ [ 1. \ 6. \ 15. \ 20. \ 15. \ 6. \ 1. ] \\
n = 7 &: \ [ 1. \ 7. \ 21. \ 35. \ 35. \ 21. \ 7. \ 1. ]
\end{align*}
\]

Exercise 8

Sieve of Eratosthenes. Write a program in Python allowing to determine all the prime numbers lower than \( N_{\text{max}} \), with an algorithm using the “sieve of Eratosthenes”.

Enter \( N_{\text{max}} : 154 \)

\[
\begin{align*}
[ & 2 \ 3 \ 5 \ 7 \ 11 \ 13 \ 17 \ 19 \ 23 \ 29 \ 31 \ 37 \ 41 \ 43 \\
& 47 \ 53 \ 59 \ 61 \ 67 \ 71 \ 73 \ 79 \ 83 \ 89 \ 97 \ 101 \ 103 \ 107 \\
& 109 \ 113 \ 127 \ 131 \ 137 \ 139 \ 149 \ 151 ]
\end{align*}
\]

4 Graphic with matplotlib

If not already done, we recommend to configure the Spyder environment to get the graphics in a separate window, as indicated in section 1.6 or as recalled herein after.

1. go into the menu: Tools/Preferences/IPython console
2. select Graphics on the right side of the window menu,
3. in Graphics backend choose Automatic (if not),
4. re-launch Spyder

```
"""""
matplot (graphic library)
matplot . pyplot is a collection of command style functions
that make matplot work like MATLAB
http://matplotlib.org/users/pyplot_tutorial.html

MATLAB , and pyplot , have the concept of the current figure
and the current axes.
All plotting commands apply to the current axes.
The function gca() returns the current axes, and gcf()
returns the current figure.
Normally, you don’t have to worry about this, because
it is all taken care of behind the scenes.
"""""

4.1 2D plotting

```
"""""
--> 2D plotting : (from file f06LibrariesA)
From Matplotlib pp30-31
"""""
import numpy as np
import matplotlib.pyplot as plt

plt.xlabel('x-axis: abscissas')
plt.ylabel('y-axis: ordinates')
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()

# evenly sampled time at 200ms intervals
r = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()

plt.clf() # clear the current figure
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()

# Clear the current axis
plt.cla()
plt.clf() # Clear the current figure
plt.close() # Close the current figure window
```

4.2 MultiFig and subplot

""" ----------------------------------------
--> MultiFig and subplot :
-------------------------------------------
"""
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

# plt.figure(1) # optional (page 34)
plt.subplot(211) # subplot(nrows, ncols, plot_number)
plt.plot(t1, f(t1), 'ro', t2, f(t2), 'g')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show() # optional here

plt.subplot(211) # becomes the current figure
plt.plot(t2, np.sin(2*np.pi*t2), 'b:')

# A figure with a 2x3 grid of Axes (page 385):
fig, ax_lst = plt.subplots(2, 3)
plt.subplot(235)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'c')

Exercise 9

Write a Python program to draw the Lissajous curves \( t \in [0, 2\pi] \mapsto (\sin(at), \sin(bt)) \), as shown in the figure below. For this purpose, create the file exoLissajousTools.py given below and create a second Python file exoLissajousDrawing.py using the function "sinatsinbt".

# File exof41LissajousTools.py
import numpy as np

def sinatsinbt(t,a,b):
    x = np.sin(a*t)
    y = np.sin(b*t)
4.3 Working with text

"""
--> Working with text (p36)
The text() command can be used to add text in an arbitrary location
The xlabel(), ylabel() and title() are used to add text in the indicated locations
"""

```python
import numpy as np
import matplotlib.pyplot as plt

plt.figure('This is a new figure')
# Normal distribution:
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000) # Normal distribution
# the histogram of the data:
N, bins, patches = plt.hist(x, bins=100, normed=1, facecolor='g', alpha=0.75)

# bins : number of bars (classes)
# alpha: transparency
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'\$\mu=100, \ \sigma=15\$') # Latex
plt.axis([40, 160, 0, 0.03])
```
plt.grid(True)

# Standard normal distribution:
plt.figure(2)
mu, sigma = 0, 1
x = mu + sigma * np.random.randn(10000)
plt.hist(x, bins=50, normed=1, facecolor='c', alpha=0.75)
t = np.linspace(-4, 4, 200)
yt = np.exp(-t**2/2) / (np.sqrt(2*np.pi))  # the "bell curve"
plt.plot(t, yt, 'b--', linewidth=2)
plt.text(1.05, .30, 'bell curve', color='b', fontsize=18)
plt.text(1.05, .25, r'$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$', color='b', fontsize=20)
plt.title('Standard normal distribution', fontsize=24, color='blue')
plt.legend(['the bell curve'], loc='best')
plt.axis([-4.2, 4.2, 0.0, 0.45])

4.4 Drawing a circle - axis equal

""" ----------------------------------------
--> drawing a circle
------------------------------------------ """
import numpy as np
import matplotlib.pyplot as plt
theta = np.linspace(0, 2*np.pi, 40)
x = np.cos(theta)
y = np.sin(theta)
plt.figure()
plt.plot(x, y)  # it doesn't look like a circle
plt.axis("equal")  # now, it looks like a circle
4.5 3D curve plotting

```python
""" ----------------------------------------
--> 3D curve plotting : (p2043)
---------------------------------------- """

from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

nbpts = 400
theta = np.linspace(-12 * np.pi, 12 * np.pi, nbpts)
z = np.linspace(-2, 2, nbpts)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)

# ---> solution 1 :
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(x, y, z, label='parametric 3D curve')
ax.legend()

# ---> solution 2:
plt.gca(projection='3d').plot(x,y,z,label='parametric 3D curve')
plt.legend()

# ---> solution 3 (multiplotting):
fig = plt.figure()
ax = fig.add_subplot(234, projection='3d')
ax.plot(x, y, z, label='green 3D curve', color='green')
plt.legend()
ax = fig.add_subplot(233, projection='3d')
ax.plot(x, y, z, label='red 3D curve', color='r')
plt.legend()
```
4.6 Surface plotting

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm  # provides a large set of colormaps
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3d

X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + 2*Y**2)
Z = np.sin(R)
fig = plt.figure(2)
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z,
                       rstride=1, cstride=1, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
fig.colorbar(surf, shrink=0.5, aspect=5)
```

---

**Surface plotting**

```
""" ----------------------------------------
--> Surface plotting : (p2057)
------------------------------------------- """
```
"""
 --> Sphere : p2058
 """

import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = 10 * np.outer(np.cos(u), np.sin(v))
y = 10 * np.outer(np.sin(u), np.sin(v))
z = 10 * np.outer(np.ones(np.size(u)), np.cos(v))

# Display 1 (rstride & cstride control the display sample):
ax.plot_surface(x, y, z, rstride=2, cstride=2, color='w')
# Display 2 (with transparency controlled by alpha):
ax.plot_surface(x, y, z, rstride=4, cstride=4, color='w', alpha=0.50)
# Display 3:
ax.plot_wireframe(x, y, z, rstride=4, cstride=4, color="g")
import numpy as np
import matplotlib.pyplot as plt

# Generate torus mesh
angle = np.linspace(0, 2 * np.pi, 32)
theta, phi = np.meshgrid(angle, angle)
r, R = .25, 1.
X = (R + r * np.cos(phi)) * np.cos(theta)
Y = (R + r * np.cos(phi)) * np.sin(theta)
Z = r * np.sin(phi)

# Display the mesh
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.set_xlim3d(-1, 1)
ax.set_ylim3d(-1, 1)
ax.set_zlim3d(-1, 1)
ax.plot_surface(X, Y, Z, color='w', rstride=1, cstride=1)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
a = 5   # main radius
b = 1   # smaller radius
k = 2
u = np.linspace(0, 2 * np.pi, 200)
v = np.linspace(0, 2 * np.pi, 200)
x = np.outer(a + b * np.cos(v), np.cos(u))
y = np.outer(a + b * np.cos(v), np.sin(u))
z = b * np.outer(np.sin(v), np.cos(k * u))
ax.plot_surface(x, y, z, rstride=4, cstride=4, color='c')

PARAMETRIC SURFACE : a polynomial example
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
u = np.linspace(-1, 1, 100)
v = np.linspace(-1, 1, 100)
x = np.outer(1 + u - u**3, v - v**2 + v**3)
y = np.outer(u + u**3, 1 - v + 2 * v**2 + v**3)
z = np.outer(-1 + 2*u + u**2, 2 * v + 4 * v**2 - v**3)
ax.plot_surface(x, y, z, rstride=4, cstride=4, color='c')
### 4.7 Level set

```python
fig = plt.figure(1, figsize=(16,6))
ax = fig.add_subplot(121, projection='3d')
X = np.arange(-1, 1, 0.02)
Y = np.arange(-1, 1, 0.02)
X, Y = np.meshgrid(X, Y)
R = 5*X**3 + 7*Y**2 - 2*X
Z = np.sin(R) - 5 * np.cos(R)
surf = ax.plot_surface(X, Y, Z,
    rstride=1, cstride=1, cmap=cm.coolwarm,
    linewidth=0, antialiased=False)
ax = fig.add_subplot(1,2,2)
cont = plt.contour(Z, np.arange(-7,7)*1.5)
plt.clabel(cont, fmt='%d')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Level sets')
```

### 4.8 Vector fields

```python
---> streamplot : p1996
-------------------------------''''
''''
Demo of the 'streamplot' function.
A streamplot, or streamline plot, is used to display 2D vector fields.
This example shows a few features of the stream plot function:
* Varying the color along a streamline.
* Varying the density of streamlines.
* Varying the line width along a stream line.
''''
import numpy as np
import matplotlib.pyplot as plt

Y, X = np.mgrid[-3:3:100j, -3:3:100j]
U = -1 - X**2 + Y
V = 1 + X - Y**2

fig0, ax0 = plt.subplots()
strm = ax0.streamplot(X, Y, U, V,
                      color=U, linewidth=2, cmap=plt.cm.autumn)
fig0.colorbar(strm.lines)

fig1, (ax1, ax2) = plt.subplots(ncols=2)
ax1.streamplot(X, Y, U, V, density=[0.5, 1])
speed = np.sqrt(U**2 + V**2)
lw = 5*speed / speed.max()
ax2.streamplot(X, Y, U, V,
               density=0.6, color='k', linewidth=lw)

4.9 Mouse acquisition

import numpy as np
import matplotlib.pyplot as plt

def AcquisitionPolygone(color1,color2) :
    #i = 0;
    x = []
    y = []
    coord = 0
    while coord != []:
        coord = plt.ginput(1, mouse_add=1, mouse_stop=3,
                           mouse_pop=2)
        if coord != []:
            xx = coord[0][0]
            yy = coord[0][1]
```python
plt.plot(xx, yy, color1, ms=8);
x.append(xx);
y.append(yy);
plt.draw()
if len(x) > 1:
    plt.plot([x[-2], x[-1]], [y[-2], y[-1]], color2)
return x, y
plt.cla()  # Clear axis
fig = plt.figure(1, (8, 8))
ax = fig.add_subplot(111)
minmax = 10;
ax.set_xlim((-minmax, minmax))
ax.set_ylim((-minmax, minmax))
xi, yi = AcquisitionPolygone('or', ':r')```

```