Introduction to programmation using the Python language

Rémi Marchal
Inorganic Theoretical Chemistry group, ISCR, Rennes
remi.marchal@univ-rennes1.fr
January, 28th 2019

First hints

Outline

Outline

Outline

A bit of history

1991 : Guido Van Rossum starts to develop the Python programming language.
2001 : Creation of the Python Software Foundation, a non-profit organization aiming at foster development of the Python community.
2009 : Creation of Python3.

Outline

What is python ?

Quesaco ?

What is python ?

Quesaco ?

- Python is an Object-oriented language. Is allows you to create and manipulate easily some objects

What is python ?

Quesaco ?

- Python is an Object-oriented language. Is allows you to create and manipulate easily some objects
- It is an interpreted language, so no compilation needed

Outline

Why Python ?

Advantages

There is several advantages in using Python

Why Python ?

Advantages

There is several advantages in using Python

- Strong developer community so a lot of libraries ready to be used

Why Python ?

Advantages

There is several advantages in using Python

- Strong developer community so a lot of libraries ready to be used
- A large user community so a lot of forum and tutorial available on the web

Why Python ?

Advantages

There is several advantages in using Python

- Strong developer community so a lot of libraries ready to be used
- A large user community so a lot of forum and tutorial available on the web
- Several graphical libraries available

2 ways to write program in python

Through the interpreter

```
pr075014:examples rmarchal$ python
Python 2.7.14 (default, Sep 22 2017, 00:06:07)
[GCC 4.2.1 Compatible Apple LLVM 8.1.0 (clang-802.0.42)
] on darwin
Type "help", "copyright", "credits" or "license" for mo
re information.
>>> a=3.5
>>> b=2.0
>>> c=a*b
>>> print c
7.0
>>> quit()
pr075014:examples rmarchal$
```

2 ways to write program in python

Through a script

```
pr075014:examples rmarchal$ cat test.py
a=3.5
b=2.0
c=a*b
print c
pr075014:examples rmarchal$ python test.py
7.0
pr075014:examples rmarchal$
```

2 ways to write program in python

Which way to choose ?

2 ways to write program in python

Which way to choose ?

It depends what you want to do.

2 ways to write program in python

Which way to choose ?

It depends what you want to do.

- If you just want to do a quite small calculation, you should use the interpreter

2 ways to write program in python

Which way to choose ?

It depends what you want to do.

- If you just want to do a quite small calculation, you should use the interpreter
- If you want to write a quite long program, use the script

2 ways to write program in python

Which way to choose ?

It depends what you want to do.

- If you just want to do a quite small calculation, you should use the interpreter
- If you want to write a quite long program, use the script
- If you want to write a program and execute it several times, use the script

Outline

Basics rules

What to avoid ?

When writing a program in python, there is some basics rules that you should follow :

- Take care about the indentation
- Avoid infinite loops
- THINK BEFORE CODING

Outline

Outline

What is a variable and why to use it ?

What is a variable ?

What is a variable and why to use it ?

What is a variable ?

- A variable is a little place in the Memory of our program and of your computer where you code should be able to store data.
  You can model your computer as a cabinet with a lot of drawer, each of them able to store a variable. When you need to access a variable, you just need to open the proper drawer

What is a variable and why to use it ?

What is a variable ?

- A variable is a little place in the Memory of our program and of your computer where you code should be able to store data.
  You can model your computer as a cabinet with a lot of drawer, each of them able to store a variable. When you need to access a variable, you just need to open the proper drawer
- Indeed, it is great to be able to make some mathematical operation but if you are not able to store the result somewhere it is useless.

What is a variable and why to use it ?

What is a variable ?

- A variable is a little place in the Memory of our program and of your computer where you code should be able to store data.
  You can model your computer as a cabinet with a lot of drawer, each of them able to store a variable. When you need to access a variable, you just need to open the proper drawer
- Indeed, it is great to be able to make some mathematical operation but if you are not able to store the result somewhere it is useless.

So no way to write a code without variables

Outline

How to create and use variables in Python ?

How to create a variable and address a value ?

How to create and use variables in Python ?

How to create a variable and address a value ?

In the Python language, it is something extremely simple to do.

How to create and use variables in Python ?

How to create a variable and address a value ?

In the Python language, it is something extremely simple to do.
Indeed, you just have to write something like this :
variable_name=value

How to create and use variables in Python ?

How to create a variable and address a value ?

In the Python language, it is something extremely simple to do.
Indeed, you just have to write something like this :
variable_name=value

Example

Let assume that you want to create a variable named *nb* in which you want to store the value 5. Thus, you just have to write this :
nb=5

Outline

The different types of variables

Variable types

Basically, there is 3 main type of variables :

- The integer (*int* in Python)
- The floating ponts (*float* in Python)
- The string (*str* in Python)

Outline

Outline

What is a function ?

A function is a group of instruction that allows you to achieve specific tasks.
To call them you just have to use the following syntax :
name_of_the_function(parameters)
It exists several intrinsic functions in Python, some of them discussed below.

Outline

The *print* function

print

This function allows you to display either some text or the value of a variable on the screen.
The syntax of this function is the following :
print(what_you_want_to_print)

The *print* function

print

This function allows you to display either some text or the value of a variable on the screen.
The syntax of this function is the following :
print(what_you_want_to_print)

Example

Python script

```
a=5.3
b=1.2
c=a+b
print(a,'+',b,'=',c)
```

The *print* function

print

This function allows you to display either some text or the value of a variable on the screen.
The syntax of this function is the following :
print(what_you_want_to_print)

Example

Python script

```
a=5.3
b=1.2
c=a+b
print(a,'+',b,'=',c)
```

Result

$5.3 + 1.2 = 6.5$

The *type* function

type

This function returns the type of a variable.
Its syntax is :
type(nom_of_the_variable)

The *type* function

type

This function returns the type of a variable.
Its syntax is :
type(nom_of_the_variable)

Example

Python script

```
string='text'
floating=4.5
integer=4
print(type(string))
print(type(floating))
print(type(integer))
```

The *type* function

type

This function returns the type of a variable.
Its syntax is :
type(nom_of_the_variable)

Example

Python script

```
string='text'
floating=4.5
integer=4
print(type(string))
print(type(floating))
print(type(integer))
```

Result

```
<class 'str'>
<class 'float'>
<class 'int'>
```

variable type conversions

Convert a variable to floating point

This can be done using the *float()* Python function.

variable type conversions

Convert a variable to floating point

This can be done using the *float()* Python function.

Example

Python script

```
integer=4
string='5'
floating=float(integer)
print(floating)
floating=float(string)
print(floating)
```

variable type conversions

Convert a variable to floating point

This can be done using the *float()* Python function.

Example

Python script

```
integer=4
string='5'
floating=float(integer)
print(floating)
floating=float(string)
print(floating)
```

Result

```
4.0
5.0
```

variable type conversions

Convert a variable to an integer

This can be done using the *int()* Python function.

variable type conversions

Convert a variable to an integer

This can be done using the *int()* Python function.

Example

Python script

```
floating=4.9
string='5'
integer=int(floating)
print(integer)
integer=int(string)
print(floating)
```

variable type conversions

Convert a variable to an integer

This can be done using the *int()* Python function.

Example

Python script

```
floating=4.9
string='5'
integer=int(floating)
print(integer)
integer=int(string)
print(floating)
```

Result

```
4
5
```

variable type conversions

Convert a variable to a string

This can be done using the *str()* Python function.

variable type conversions

Convert a variable to a string

This can be done using the *str()* Python function.

Example

Python script

```
floating=4.9
integer=5
string=str(floating)
print(string)
string=str(integer)
print(string)
```

variable type conversions

Convert a variable to a string

This can be done using the *str()* Python function.

Example

Python script

```
floating=4.9
integer=5
string=str(floating)
print(string)
string=str(integer)
print(string)
```

Result

```
4.9
5
```

Store keyboard entries into a variable

keyboard entry

This can be done using the *input()* Python function.
ATTENTION : this keyboard entry will be interpreted as a string

Store keyboard entries into a variable

keyboard entry

This can be done using the *input()* Python function.
ATTENTION : this keyboard entry will be interpreted as a string

Example

Python script

```
a=input("enter a value")
```

Store keyboard entries into a variable

keyboard entry

This can be done using the *input()* Python function.
ATTENTION : this keyboard entry will be interpreted as a string

Example

Python script

```
a=input("enter a value")
```

Result

```
enter a value 2.5
```

Store keyboard entries into a variable

keyboard entry

This can be done using the *input()* Python function.
ATTENTION : this keyboard entry will be interpreted as a string

Example

Python script

```
a=input("enter a value")
a=float(a)
print("a square= ",a**2)
```

Result

```
enter a value 2.5
```

Store keyboard entries into a variable

keyboard entry

This can be done using the *input()* Python function.
ATTENTION : this keyboard entry will be interpreted as a string

Example

Python script

```
a=input("enter a value")
a=float(a)
print("a square= ",a**2)
```

Result

```
enter a value 2.5
a square= 6.25
```

Outline

Outline

What is a module ?

Definition of a module

A module is basically a piece of code that have been encapsulated in a file. The variables and functions that are related to this module are also encapsulated in the file.
Thus, one have just to call the module in order to use its functionalities.

Why to use modules ?

There is a lot of different modules that have been developed in Python and it is quite easy to install and use them.

Outline

How to import modules ?

Various methods

It exists basically 3 ways to import modules

1. The basic *import* method
2. Import a module and rename it
3. The partial import method

How to import modules ?

The basic *import* method

To import a module, you can use the following syntax in your python script :
import module_name
Using this method, all the functions included in the module are imported. To use one of them, you have to use the following syntax :
module_name.function_name

How to import modules ?

The basic *import* method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

How to import modules ?

The basic *import* method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
a=0.0
res=exp(a)
```

How to import modules ?

The basic *import* method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
a=0.0
res=exp(a)
```

Result

```
Traceback (most recent
call last) :
File "t.py", line 2, in
<module>
res=exp(a)
NameError : name 'exp'
is not defined
```

How to import modules ?

The basic *import* method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
a=0.0
res=exp(a)
import math
res=math.exp(a)
print(res)
```

Result

```
Traceback (most recent
call last) :
File "t.py", line 2, in
<module>
res=exp(a)
NameError : name 'exp'
is not defined
```

How to import modules ?

The basic *import* method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
a=0.0
res=exp(a)
import math
res=math.exp(a)
print(res)
```

Result

```
Traceback (most recent
call last) :
File "t.py", line 2, in
<module>
res=exp(a)
NameError : name 'exp'
is not defined
1.0
```

How to import modules ?

Import a module and rename it

This method is more or less the same as the basic method. However, you can change the name of the module using the following syntax
import module_name as new_name

How to import modules ?

Import a module and rename it : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

How to import modules ?

Import a module and rename it : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
import math as ma
a=0.0
res=ma.exp(a)
print(res)
```

How to import modules ?

Import a module and rename it : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
import math as ma
a=0.0
res=ma.exp(a)
print(res)
```

Result

```
1.0
```

How to import modules ?

The partial import method

Instead of importing the whole module, you can import only a specific function of it using the following syntax :

from module_name import function_name

For example, if you only need the *exp* function of the *math* module, you can do :

from math import exp

How to import modules ?

The partial import method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

How to import modules ?

The partial import method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
from math import exp
a=0.0
res=exp(a)
print(res)
```

How to import modules ?

The partial import method : Example

Lets try to compute the exponential of 0.0. The exponential function is included in the *math* module.

Python script

```
from math import exp
a=0.0
res=exp(a)
print(res)
```

Result

```
1.0
```

Conditional statements and loops

Outline

Outline

What is conditional statements ?

Definition

Conditional statements is something widely used. Indeed, it allows you to execute a part of the code only under specific conditions

What is conditional statements ?

Definition

Conditional statements is something widely used. Indeed, it allows you to execute a part of the code only under specific conditions

Conditional operators

symbol	signification
$<$	Strictly inferior
$>$	Strictly superior
$\leqslant$	Inferior or equal
$\geqslant$	Superior or equal
$==$	equal
$!=$	different

Outline

Indentation

What is indentation ?

After writing your conditional statement, one should introduce spaces at the beginning of each line in the part of the code that should be executed if the condition is fulfill. This amount of space should be the same for each lines
This is what we call indentation.

Indentation

What is indentation ?

After writing your conditional statement, one should introduce spaces at the beginning of each line in the part of the code that should be executed if the condition is fulfill. This amount of space should be the same for each lines
This is what we call indentation.

Example with algorithm

a=3
b=2
If a > b
    write a is higher than b
    write b is lower than a
Here, the last 2 lines have been indented (adding space at the beginning)

Outline

The minimal conditional statement (*if*)

The simple *if* statement

This allows you to execute some instructions only if a condition if fulfill.

The syntax is the following :

if condition :

What to do if the condition is fulfill

The minimal conditional statement (*if*)

The simple *if* statement

This allows you to execute some instructions only if a condition if fulfill.

The syntax is the following :

if condition :

What to do if the condition is fulfill

Python script

```
a=4.2
if a>0 :
    print("a is positive")
```

The minimal conditional statement (*if*)

The simple *if* statement

This allows you to execute some instructions only if a condition if fulfill.

The syntax is the following :

if condition :

What to do if the condition is fulfill

Python script

```
a=4.2
if a>0 :
    print("a is positive")
```

Result

```
a is positive
```

The complete conditional (*if, elif, else*)

The complete conditional statement

This allows you to execute some instructions only if a condition if fulfill and another part if the condition is not fulfill .

The syntax is the following :

if condition :

    What to do if the condition is fulfill

else :

    What to do if the condition is not fulfill

It is also possible to introduce other conditions using elif.

The complete conditional (*if*, *elif*, *else*)

Example

Python script

```
a=4.2
if a>0 :
    print("a is positive")
elif a==0 :
    print("a is zero")
else :
    print("a is negative")
```

The complete conditional (*if*, *elif*, *else*)

Example

Python script

```
a=4.2
if a>0 :
    print("a is positive")
elif a==0 :
    print("a is zero")
else :
    print("a is negative")
```

Result

```
a is positive
```

Outline

Outline

What is a loop ?

Definition of a loop

Loop is an highly important concept in programming. Indded, it allows you to repeat some operations a given amount of time.

What is a loop ?

Definition of a loop

Loop is an highly important concept in programming. Indded, it allows you to repeat some operations a given amount of time.

Loop types

It exists 2 main type of loops :

1. Repeating loops
2. Conditional loops

Outline

Repeating loops

Why and how ?

This kind of loop is what I will call the classical one.
It allows you to repeat given operations from a starting value to an end value with specific increments.
The syntax is the following :

for i in range(starting_i,last_i,increment) :
    What to repeat

The last value of a loop is exclusive (will not be used).
If you don't specify the value for the increment, it will use the default one which is 1.

ATTENTION : Don't forget the indentation

Repeating loops

Repeating loops : First example

Python script

```
for i in range(0,6) :
    print(i)
```

Repeating loops

Repeating loops : First example

Python script

```
for i in range(0,6) :
    print(i)
```

Result

```
0
1
2
3
4
5
```

Repeating loops

Repeating loops : First example

Python script

```
for i in range(0,6) :
    print(i)
```

Result

```
0
1
2
3
4
5
```

Repeating loops : Second example

Python script

```
for i in range(0,6,2) :
    print(i)
```

Repeating loops

Repeating loops : First example

Python script

```
for i in range(0,6) :
    print(i)
```

Result

```
0
1
2
3
4
5
```

Repeating loops : Second example

Python script

```
for i in range(0,6,2) :
    print(i)
```

Result

```
0
2
4
```

Outline

Conditional loops

The *while* loop

The aim of this loop is to repeat a part of the code until a given condition is fulfill.
The syntax is the following :
while condition :
    What_to_do
ATTENTION : If the condition is never fulfill, you will end up with an infinite loop and the program will never stop

Conditional loops

The *while* loop

The aim of this loop is to repeat a part of the code until a given condition is fulfill.
The syntax is the following :
while condition :
    What_to_do
ATTENTION : If the condition is never fulfill, you will end up with an infinite loop and the program will never stop

Conditional loops : example

Python script

```
a=0
while a<4 :
    print(i)
    a=a+1
```

Conditional loops

The *while* loop

The aim of this loop is to repeat a part of the code until a given condition is fulfill.
The syntax is the following :

while condition :
    What_to_do

ATTENTION : If the condition is never fulfill, you will end up with an infinite loop and the program will never stop

Conditional loops : example

Python script

```
a=0
while a<4 :
    print(i)
    a=a+1
```

Result

```
0
1
2
3
```

Outline

Stop a loop

The *break* statement

This statement aims at stopping a loop.
The syntax is the following (assuming you are inside a loop) :
break

Stop a loop

The *break* statement

This statement aims at stopping a loop.
The syntax is the following (assuming you are inside a loop) :
break

break : example

Python script

```
for i in range(0,1000,2) :
    print(i)
    if i>10 :
        break
```

Stop a loop

The *break* statement

This statement aims at stopping a loop.
The syntax is the following (assuming you are inside a loop) :
break

break : example

Python script

```
for i in range(0,1000,2) :
    print(i)
    if i>10 :
        break
```

Result

```
0
2
4
6
8
10
12
```

The notion of lists in Python

Outline

What is a list ?

Definition

Lists in Python is something essential. Indeed, they are used to store several data in the same variable. This variable is then called a list and contains several elements, each of them characterized by a list-index. It is also possible to create list of lists.

Outline

How a define a list ?

Syntax

There is 2 ways to define an empty list :

- Either using the syntax :*variable=list()*
- Or this one : *variable=[ ]*

Outline

List Rules

Rules

1. The first index of a list is always 0
2. A given element of a list should be called with "[ ]". For example, if you want to access the 2nd element of a list called *list1*, you should call it like this : *list1[1]* (remember, the index of the first element is 0)
3. Don't try to make additions directly of 2 lists (don't write list3=list2+list1). If you want to do it, it is safer to make additions elements by elements
4. You can easily know the size (number of elements) of a list using the len statement (for example, *len(list1)* will render the number of elements of the list *list1*)

Outline

Outline

create a list with predefined values

How to do it ?

To create a list from an already known set of values, the syntax is the following :
list_name=[values_seperated_by_commas]

create a list with predefined values

How to do it ?

To create a list from an already known set of values, the syntax is the following :
list_name=[values_seperated_by_commas]

Example

Python script

```
list1=[0.1,0.2,0.3]
print(list1)
```

create a list with predefined values

How to do it ?

To create a list from an already known set of values, the syntax is the following :
list_name=[values_seperated_by_commas]

Example

Python script

```
list1=[0.1,0.2,0.3]
print(list1)
```

Result

```
[0.1,0.2,0.3]
```

Outline

create an empty list

How to do it ?

To create an empty list, the widely used syntax is the following :

list_name=[ ]

create an empty list

How to do it ?

To create an empty list, the widely used syntax is the following :

list_name=[ ]

Example

Python script

```
list1=[ ]
print(list1)
```

create an empty list

How to do it ?

To create an empty list, the widely used syntax is the following :

list_name=[ ]

Example

Python script

```
list1=[ ]
print(list1)
```

Result

```
[ ]
```

Outline

Outline

Adding elements at the end of a list

How to do it ?

This can be done by appending an existing list. The syntax is the following :
list_name.append(new_value_to_add_at_the_end)

Adding elements at the end of a list

How to do it ?

This can be done by appending an existing list. The syntax is the following :
list_name.append(new_value_to_add_at_the_end)

Example

Python script

```
list1=[ ]
for i in range(0,4) :
    list1.append(i)
print(list1)
```

Adding elements at the end of a list

How to do it ?

This can be done by appending an existing list. The syntax is the following :
list_name.append(new_value_to_add_at_the_end)

Example

Python script

```
list1=[ ]
for i in range(0,4) :
    list1.append(i)
print(list1)
```

Result

[0,1,2,3]

Adding elements at the end of a list

How to do it ?

This can be done by appending an existing list. The syntax is the following :
list_name.append(new_value_to_add_at_the_end)

Example

Python script

```
list1=[ ]
for i in range(0,4) :
    list1.append(i)
print(list1)
list1.append(4)
print(list1)
```

Result

[0,1,2,3]

Adding elements at the end of a list

How to do it ?

This can be done by appending an existing list. The syntax is the following :
list_name.append(new_value_to_add_at_the_end)

Example

Python script

```
list1=[ ]
for i in range(0,4) :
    list1.append(i)
print(list1)
list1.append(4)
print(list1)
```

Result

```
[0,1,2,3]
[0,1,2,3,4]
```

Outline

Insert an element at a given position in the list

How to do it ?

To do it, one should use the *insert* function. The syntax is the following :
list_name.insert(position,value)

Insert an element at a given position in the list

How to do it ?

To do it, one should use the *insert* function. The syntax is the following :
list_name.insert(position,value)

Example

Python script

```
a=[0,1,2,4]
print(a)
```

Insert an element at a given position in the list

How to do it ?

To do it, one should use the *insert* function. The syntax is the following :
list_name.insert(position,value)

Example

Python script

```
a=[0,1,2,4]
print(a)
```

Result

```
[0,1,2,4]
```

Insert an element at a given position in the list

How to do it ?

To do it, one should use the *insert* function. The syntax is the following :
list_name.insert(position,value)

Example

Python script

```
a=[0,1,2,4]
print(a)
a.insert(3,3.0)
print(a)
```

Result

```
[0,1,2,4]
```

Insert an element at a given position in the list

How to do it ?

To do it, one should use the *insert* function. The syntax is the following :
list_name.insert(position,value)

Example

Python script

```
a=[0,1,2,4]
print(a)
a.insert(3,3.0)
print(a)
```

Result

```
[0,1,2,4]
[0,1,2,3.0,4]
```

Outline

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

Example extend method

Python script

```
list1=[0,1,2,3]
list2=[4,5,6,7]
list1.extend(list2)
print(list1)
```

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

Example extend method

Python script

```
list1=[0,1,2,3]
list2=[4,5,6,7]
list1.extend(list2)
print(list1)
```

Result

```
[0,1,2,3,4,5,6,7]
```

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

Example "+" method

Python script

```
list1=[0,1,2,3]
list2=[4,5,6,7]
list3=list1+list2
print(list3)
```

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

Example "+" method

Python script

```
list1=[0,1,2,3]
list2=[4,5,6,7]
list3=list1+list2
print(list3)
```

Result

```
[0,1,2,3,4,5,6,7]
```

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

Example "+=" method

Python script

```
list1=[0,1,2,3]
list2=[4,5,6,7]
list1+=listt2
print(list1)
```

List concatenation

How to do it ?

There is 3 ways to it :

1. The extend method :
   list1.extend(list2)
2. The "+" method :
   list1=list1+list2
3. The "+=" method :
   list1+=list2

Example "+=" method

Python script

```
list1=[0,1,2,3]
list2=[4,5,6,7]
list1+=listt2
print(list1)
```

Result

```
[0,1,2,3,4,5,6,7]
```

Outline

Remove an element of a list

How to do it ?

There is 2 ways to it depending if you want to remove an index or a value :

- If you want to remove an index :
  del list_name[index_to_remove]
- If you want to remove a value :
  list_name.remove(value_to_remove)
  It will only remove the value one time if it appears several times in the list.

Remove an element of a list

Remove an element of a list

Example with *del*

Python script

```
list1=[0,1,2,3,1]
list1.del[2]
print(list1)
```

Remove an element of a list

Example with *del*

Python script

```
list1=[0,1,2,3,1]
list1.del[2]
print(list1)
```

Result

```
[0,1,3,1]
```

Remove an element of a list

Example with *del*

Python script

```
list1=[0,1,2,3,1]
list1.del[2]
print(list1)
```

Result

```
[0,1,3,1]
```

Example with *remove*

Python script

```
list1=[0,1,2,3,1]
list1.remove(1)
print(list1)
```

Remove an element of a list

Example with *del*

Python script

```
list1=[0,1,2,3,1]
list1.del[2]
print(list1)
```

Result

```
[0,1,3,1]
```

Example with *remove*

Python script

```
list1=[0,1,2,3,1]
list1.remove(1)
print(list1)
```

Result

```
[0,2,3,1]
```

Outline

Browse a list

How to do it ?

There is 2 ways to it :

- Using a loop starting from index 0 to the size of the list
- Using a loop over all the list elements

Browse a list

How to do it ?

There is 2 ways to it :

- Using a loop starting from index 0 to the size of the list
- Using a loop over all the list elements

Example : index loop

Python script

```
list1=[0,1,2]
for i in range(0,len(list1)) :
    print(list1[i])
```

Browse a list

How to do it ?

There is 2 ways to it :

- Using a loop starting from index 0 to the size of the list
- Using a loop over all the list elements

Example : index loop

Python script

```
list1=[0,1,2]
for i in range(0,len(list1)) :
    print(list1[i])
```

Result

```
0
1
2
```

Browse a list

How to do it ?

There is 2 ways to it :

- Using a loop starting from index 0 to the size of the list
- Using a loop over all the list elements

Browse a list

How to do it ?

There is 2 ways to it :

- Using a loop starting from index 0 to the size of the list
- Using a loop over all the list elements

Example : loop over list elements

Python script

```
list1=[0,1,2]
for i in list1 :
    print(list1[i])
```

Browse a list

How to do it ?

There is 2 ways to it :

- Using a loop starting from index 0 to the size of the list
- Using a loop over all the list elements

Example : loop over list elements

Python script

```
list1=[0,1,2]
for i in list1 :
    print(list1[i])
```

Result

```
0
1
2
```

Reading and writing files

Outline

Outline

Why to read files ?

Why ?

Why to read files ?

Why ?

- It is an extremely important concept in scientific programming

Why to read files ?

Why ?

- It is an extremely important concept in scientific programming
- Indeed, most of our calculation need input data and it's easier to read them from input files

Why to read files ?

Why ?

- It is an extremely important concept in scientific programming
- Indeed, most of our calculation need input data and it's easier to read them from input files
- It is also of prime importance if you aim at post-treatment program development

Introduction

Advantages in Python

Introduction

Advantages in Python

- Despite Python is an interpreted language and thus slower than compiled languages such as C or Fortran, one of the main advantage of using Python is that it's quite easy to open and read files with it.

Introduction

Advantages in Python

- Despite Python is an interpreted language and thus slower than compiled languages such as C or Fortran, one of the main advantage of using Python is that it's quite easy to open and read files with it.
- Indeed, this task is quite difficult in Fortran and C

Introduction

Advantages in Python

- Despite Python is an interpreted language and thus slower than compiled languages such as C or Fortran, one of the main advantage of using Python is that it's quite easy to open and read files with it.
- Indeed, this task is quite difficult in Fortran and C
- That's why most of the post-treatment program are written in both Python (for data extraction) and Fortran or C (for the calculation part).

Outline

Reading file in Python

Reading file in Python

Main idea

- The idea is to browse an entire file and to store its contains in a list, each of the list elements being either a word or a line.

Reading file in Python

Main idea

- The idea is to browse an entire file and to store its contains in a list, each of the list elements being either a word or a line.
- This list will be then browsed in order to extract only the informations that are relevant for the needed calculation.

Reading file in Python

Main idea

- The idea is to browse an entire file and to store its contains in a list, each of the list elements being either a word or a line.
- This list will be then browsed in order to extract only the informations that are relevant for the needed calculation.

An essential rule

Attention : Whatever the method used, reading a file will lead to string variable. Thus, one should not forget to convert them to integers or floating points before doing any calculation.

Outline

The *open* method

What is it ?

This method for reading files is based on the *open* intrinsic Python function.

The *open* method : How ?

The *modus operandi*

The *open* method : How ?

The *modus operandi*

1. Apply the function *open* with its readlines functionality to a file

The *open* method : How ?

The *modus operandi*

1. Apply the function *open* with its readlines functionality to a file
2. Store the result in a list.

First example

The "test" file

1 2 3 4 5
6 7 8 9 10
11 12 13 14

First example

The "test" file

1 2 3 4 5
6 7 8 9 10
11 12 13 14

First example

Python script

```
f=open("test").readlines()
print(f)
```

First example

The "test" file

1 2 3 4 5
6 7 8 9 10
11 12 13 14

First example

Python script

```
f=open("test").readlines()
print(f)
```

Result

[1 2 3 4 5 \n,6 7 8 9 10 \n, 11 12 13 14]

First example

The "test" file

1 2 3 4 5
6 7 8 9 10
11 12 13 14

First example

Python script

```
f=open("test").readlines()
print(f)
```

Result

[1 2 3 4 5 \n,6 7 8 9 10 \n, 11 12 13 14]

Thus, each of the list element are lines. So how to create lists with only words and not lines.

Splitting lines into words

the str.split method

We would like now to read a file and store it as a list of lists, each of the lists being a line and each of the lists elements words.
For this, we can use the str.split functionality.

str.split example

The "test" file

1 2 3 4 5
6 7 8 9 10
11 12 13 14

str.split example

The "test" file

1 2 3 4 5
6 7 8 9 10
11 12 13 14

str.split example

Python script

```
l1=[ ]
f=open("test").readlines()
# browse each line
for i in range(0,len(f)) :
    l2=[ ]
# browse each word of the line
    for j in range(0,len(str.split(f[i]))) :
        l2.append(str.split(f[i])[j])
    l1.append(l2)
```

str.split example

The "test" file

1 2 3 4 5
6 7 8 9 10
11 12 13 14

str.split example

Python script

```
l1=[ ]
f=open("test").readlines()
# browse each line
for i in range(0,len(f)) :
    l2=[ ]
# browse each word of the line
    for j in range(0,len(str.split(f[i]))) :
        l2.append(str.split(f[i])[j])
    l1.append(l2)
```

Result

[[1,2,3,4,5],
[6,7,8,9,10],
[11,12,13,14]]

Outline

Writing in a file

How ?

This is based on the use of the intrinsic *write* Python function

line break

This is done by writing : '\n'

Writing in a file

modus operandi

Writing in a file

modus operandi

1. Call the open function on a new file and the "w" format

Writing in a file

modus operandi

1. Call the open function on a new file and the "w" format
2. Write inside everything you want using the functionality *write*

Writing in a file

modus operandi

1. Call the open function on a new file and the "w" format
2. Write inside everything you want using the functionality *write*
3. Close the file with the functionality *close*

Writing files : Example

Writing files : Example

First example

Python script

```
a=[[First,line],[Second,line]]
f=open("test","w")
for i in range(0,len(a)) :
    for j in range(0,len(a[i]) :
        f.write(a[i][j])
    f.write("\n")
f.close()
```

Writing files : Example

First example

Python script

```
a=[[First,line],[Second,line]]
f=open("test","w")
for i in range(0,len(a)) :
    for j in range(0,len(a[i]) :
        f.write(a[i][j])
    f.write("\n")
f.close()
```

File "test"

```
First line
Second line
```

The Numpy module

Outline

The Numpy module : Presentation

What is it ?

This module allows you to

- Use trigonometric functions
- Make some calculations on matrices
- Solve equations systems

Outline

Numpy : Trigonometric functions

Numpy : Trigonometric functions

Which of them ?

- Cosine (numpy.cos) et Arccos (numpy.arccos)
- Sinus (numpy.sin) et Arcsin (numpy.arcsin)
- Tangent (numpy.tan) et Arctan (numpy.arctan)

Numpy : Trigonometric functions

Which of them ?

- Cosine (numpy.cos) et Arccos (numpy.arccos)
- Sinus (numpy.sin) et Arcsin (numpy.arcsin)
- Tangent (numpy.tan) et Arctan (numpy.arctan)

Attention : These functions are all in radian (It exist the function **numpy.pi** for the pi number)

Numpy : Trigonometric functions

Numpy : Trigonometric functions

Example

Python script

```
import numpy as np
a=0.0
b=np.pi/2.0
c=np.pi/3.0
print(a,b,c)
```

Numpy : Trigonometric functions

Example

Python script

```
import numpy as np
a=0.0
b=np.pi/2.0
c=np.pi/3.0
print(a,b,c)
```

Result

```
0.0 1.57 1.04
```

Numpy : Trigonometric functions

Example

Python script

```
import numpy as np
a=0.0
b=np.pi/2.0
c=np.pi/3.0
print(a,b,c)
cosa=np.cos(a)
cosb=np.cos(b)
cosc=np.cos(c)
```

Result

0.0 1.57 1.04

Numpy : Trigonometric functions

Example

Python script

```
import numpy as np
a=0.0
b=np.pi/2.0
c=np.pi/3.0
print(a,b,c)
cosa=np.cos(a)
cosb=np.cos(b)
cosc=np.cos(c)
```

Result

```
0.0 1.57 1.04
1.0 6.12e-17 0.50
```

Numpy : Trigonometric functions

Example

Python script

```
import numpy as np
a=0.0
b=np.pi/2.0
c=np.pi/3.0
print(a,b,c)
cosa=np.cos(a)
cosb=np.cos(b)
cosc=np.cos(c)
print(np.arccos(cosa))
print(np.arccos(cosb))
print(np.arccos(cosc))
```

Result

```
0.0 1.57 1.04
1.0 6.12e-17 0.50
```

Numpy : Trigonometric functions

Example

Python script

```
import numpy as np
a=0.0
b=np.pi/2.0
c=np.pi/3.0
print(a,b,c)
cosa=np.cos(a)
cosb=np.cos(b)
cosc=np.cos(c)
print(np.arccos(cosa))
print(np.arccos(cosb))
print(np.arccos(cosc))
```

Result

```
0.0 1.57 1.04
1.0 6.12e-17 0.50
0.0
1.570
1.04
```

Outline

Outline

3. Matrices

Create vectors with zeros

Create vectors with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Create vectors with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Syntax : a=numpi.zeros(m)

a will be a zeros vector of size **m**

Create vectors with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Syntax : a=numpi.zeros(m)

a will be a zeros vector of size **m**

Example

Python script

```
import numpy as np
a=np.zeros(4)
print(a)
```

Create vectors with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Syntax : a=numpi.zeros(m)

a will be a zeros vector of size **m**

Example

Python script

```
import numpy as np
a=np.zeros(4)
print(a)
```

Result

```
[0. 0. 0. 0.]
```

Create vectors with given values

Create vectors with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Create vectors with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Syntax : a=numpi.array([values]).

Each values should be separated by comas.

a will be a vector that contains the given values

Create vectors with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Syntax : a=numpi.array([values]).

Each values should be separated by comas.

a will be a vector that contains the given values

Example

Python script

```
import numpy as np
a=np.array([1.0,1.1,1.2])
print(a)
```

Create vectors with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Syntax : a=numpi.array([values]).

Each values should be separated by comas.

a will be a vector that contains the given values

Example

Python script

```
import numpy as np
a=np.array([1.0,1.1,1.2])
print(a)
```

Result

```
[1.0 1.1 1.2]
```

Create matrices with zeros

Create matrices with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Create matrices with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Syntax : a=numpi.zeros((m,n)).

a will be a zeros matrix of size **m** by **n**

Create matrices with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Syntax : a=numpi.zeros((m,n)).

a will be a zeros matrix of size **m** by **n**

Example

Python script

```
import numpy as np
a=np.zeros((2,3))
print(a)
```

Create matrices with zeros

The numpy.zeros function

This function of the Numpy module allows to create zeros floating points vectors (or matrices) of specific sizes

Syntax : a=numpi.zeros((m,n)).

a will be a zeros matrix of size **m** by **n**

Example

Python script

```
import numpy as np
a=np.zeros((2,3))
print(a)
```

Result

```
[[0. 0. 0.]
[0. 0. 0.]]
```

Create matrices with given values

Create matrices with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Create matrices with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Syntax : a=numpi.array([[values1],[values2]).

Each values should be separated by comas.

a will be a 2 line matrix containing the given values

Create matrices with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Syntax : a=numpi.array([[values1],[values2]).

Each values should be separated by comas.

a will be a 2 line matrix containing the given values

Example

Python script

```
import numpy as np
a=np.array([[1.0,1.1,1.2],
[2.0,2.1,2.2]])
print(a)
```

Create matrices with given values

The numpy.array function

This function allows to create vectors (or matrices) from an already known series of values

Syntax : a=numpi.array([[values1],[values2]).

Each values should be separated by comas.

a will be a 2 line matrix containing the given values

Example

Python script

```
import numpy as np
a=np.array([[1.0,1.1,1.2],
[2.0,2.1,2.2]])
print(a)
```

Result

```
[[1.2 1.1 1.2]
[2.0 2.1 2.2]]
```

Outline

Scalar product

Scalar product

numpy.dot

This Numpy function allows to compute scalar product between 2 vectors

Scalar product

numpy.dot

This Numpy function allows to compute scalar product between 2 vectors

Syntax

c=numpy.dot(a,b)

c will be a scalar, result of the scalar product of **a** by **b**.

Scalar product

numpy.dot

This Numpy function allows to compute scalar product between 2 vectors

Syntax

c=numpy.dot(a,b)

c will be a scalar, result of the scalar product of **a** by **b**.

Example

Python script

```
import numpy as np
a=np.array([1,2,3])
b=[4,5,6]
c=np.dot(a,b)
print(c)
```

Scalar product

numpy.dot

This Numpy function allows to compute scalar product between 2 vectors

Syntax

c=numpy.dot(a,b)

c will be a scalar, result of the scalar product of **a** by **b**.

Example

Python script

```
import numpy as np
a=np.array([1,2,3])
b=[4,5,6]
c=np.dot(a,b)
print(c)
```

Result

32

Outline

Produit matriciel

Produit matriciel

numpy.dot

This Numpy function allows also to multiply matrices.

Produit matriciel

numpy.dot

This Numpy function allows also to multiply matrices.

Syntax

c=numpy.dot(a,b).

c is a matrix that the result of the multiplication of **a** par **b**

Produit matriciel

numpy.dot

This Numpy function allows also to multiply matrices.

Syntax

c=numpy.dot(a,b).

c is a matrix that the result of the multiplication of **a** par **b**

Example

Python script

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
c=np.array([[4],[2],[1]])
print(np.dot(a,c))
```

Produit matriciel

numpy.dot

This Numpy function allows also to multiply matrices.

Syntax

c=numpy.dot(a,b).
c is a matrix that the result of the multiplication of **a** par **b**

Example

Python script

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
c=np.array([[4],[2],[1]])
print(np.dot(a,c))
```

Result

```
[[11]
[32]]
```

Outline

Transpose matrix

Transpose matrix

.T

This numpy function allows to transpose matrices.

Transpose matrix

.T

This numpy function allows to transpose matrices.

Syntax

c=a.T

c is the transpose matrix of **a**

Transpose matrix

.T

This numpy function allows to transpose matrices.

Syntax

c=a.T

c is the transpose matrix of **a**

Example

Python script

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a)
```

Transpose matrix

.T

This numpy function allows to transpose matrices.

Syntax

c=a.T

c is the transpose matrix of **a**

Example

Python script

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a)
```

Result

```
[[1 2 3]
[4 5 6]]
```

Transpose matrix

.T

This numpy function allows to transpose matrices.

Syntax

c=a.T

c is the transpose matrix of **a**

Example

Python script

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a)
print(a.T)
```

Result

```
[[1 2 3]
[4 5 6]]
```

Transpose matrix

.T

This numpy function allows to transpose matrices.

Syntax

c=a.T

c is the transpose matrix of **a**

Example

Python script

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a)
print(a.T)
```

Result

```
[[1 2 3]
[4 5 6]]
[[1 4]
[2 4]
[3 6]]
```

Outline

Matrix determinant

Matrix determinant

numpy.linalg.det

This numpy function allows to compute the determinant of a matrix.

Matrix determinant

numpy.linalg.det

This numpy function allows to compute the determinant of a matrix.

Syntax

c=numpy.linalg.det(a)

c will be the determinant of the squared matrix **a**.

Matrix determinant

numpy.linalg.det

This numpy function allows to compute the determinant of a matrix.

Syntax

c=numpy.linalg.det(a)

c will be the determinant of the squared matrix **a**.

Example

Python script

```
import numpy as np
from numpy.linalg import det
a=np.array([[1,2],[3,4]])
print(det(a))
```

Matrix determinant

numpy.linalg.det

This numpy function allows to compute the determinant of a matrix.

Syntax

c=numpy.linalg.det(a)

c will be the determinant of the squared matrix **a**.

Example

Python script

```
import numpy as np
from numpy.linalg import det
a=np.array([[1,2],[3,4]])
print(det(a))
```

Result

-2.00

Outline

Matrix inversion

Matrix inversion

numpy.linalg.inv

Tis Numpy function allows to invert squared matrix.

Matrix inversion

numpy.linalg.inv

Tis Numpy function allows to invert squared matrix.

Syntax

c=numpy.linalg.inv(a)

c will be the inverted **a** matrix.

Matrix inversion

numpy.linalg.inv

Tis Numpy function allows to invert squared matrix.

Syntax

c=numpy.linalg.inv(a)

c will be the inverted **a** matrix.

Example

Python script

```
import numpy as np
from numpy.linalg import inv
a=np.array([[1,2],[3,4]])
print(inv(a))
```

Matrix inversion

numpy.linalg.inv

Tis Numpy function allows to invert squared matrix.

Syntax

c=numpy.linalg.inv(a)

c will be the inverted **a** matrix.

Example

Python script

```
import numpy as np
from numpy.linalg import inv
a=np.array([[1,2],[3,4]])
print(inv(a))
```

Result

```
[[−2.00 1.0]
[1.5  − 0.5]]
```

Outline

Outline

4. Solve Equation system

Linear equation system

Linear equation system

numpy.linalg.solve

This Numpy function allows to solve linear equation systems.

Linear equation system

numpy.linalg.solve

This Numpy function allows to solve linear equation systems.

Syntaxe

c=numpy.linalg.solve(a,b) where **a** is the coefficient matrix and **b** a vector.

Solve Equation system

Example

Lets solve the linear equation system

$$\begin{cases} 3x + y = 9 \\ x + 2y = 8 \end{cases}$$

Solve Equation system

Example

Lets solve the linear equation system

$$\begin{cases} 3x + y = 9 \\ x + 2y = 8 \end{cases}$$

Python script

```
import numpy as np
from numpy.linalg import solve
a=np.array([[3,1],[1,2]])
b=np.array([9,8])
print(solve(a,b))
```

Solve Equation system

Example

Lets solve the linear equation system

$$\begin{cases} 3x + y = 9 \\ x + 2y = 8 \end{cases}$$

Python script

```
import numpy as np
from numpy.linalg import solve
a=np.array([[3,1],[1,2]])
b=np.array([9,8])
print(solve(a,b))
```

Result

```
[2 3]
```

Solve Equation system

Example

Lets solve the linear equation system

$$\begin{cases} 3x + y = 9 \\ x + 2y = 8 \end{cases}$$

Python script

```
import numpy as np
from numpy.linalg import solve
a=np.array([[3,1],[1,2]])
b=np.array([9,8])
print(solve(a,b))
```

Result

```
[2 3]
```

Solutions are x=2 and y=3

Outline

Matrix diagonalization

Matrix diagonalization

numpy.linalg.eig

This Numpy function will output the eigenvalues and eigenvectors of a matrix

Matrix diagonalization

numpy.linalg.eig

This Numpy function will output the eigenvalues and eigenvectors of a matrix

Syntax

E, V=numpy.linalg.eig(a)

E will contain the eigenvalues of **a** and **V** its eigenvectors.

Matrix diagonalization

Example

Python script

```
import numpy as np
from numpy.linalg import eig
A=np.array([[1,1,-2],[-1,2,1],[0,1,-1]])
D, V = eig(A)
print(D)
```

Matrix diagonalization

Example

Python script

```
import numpy as np
from numpy.linalg import eig
A=np.array([[1,1,-2],[-1,2,1],[0,1,-1]])
D, V = eig(A)
print(D)
```

Result

```
[2. 1. -1.]
```

Matrix diagonalization

Example

Python script

```
import numpy as np
from numpy.linalg import eig
A=np.array([[1,1,-2],[-1,2,1],[0,1,-1]])
D, V = eig(A)
print(D)
```

Result

```
[2. 1. -1.]
```

Matrix diagonalization

Example

Python script

```
import numpy as np
from numpy.linalg import eig
A=np.array([[1,1,-2],[-1,2,1],[0,1,-1]])
D, V = eig(A)
print(D)
```

Result

[2. 1. -1.]
[[3.015e − 01 − 8.017e − 01 7.071e − 01]
[9.045e − 01 − 5.345e − 01 2.435e − 17]
[3.015e − 01 − 2.672e − 01 7.071e − 01]]

Graphical representation
The matplotlib module

Outline

Outline

Definition

Matplotlib : What is it ?

Matplotlib is a Python module which, combined with the Numpy and Scipy module, is an extremely powerful tools for plotting and visualizing data .
The Matplotlib module have several advantages :

- free of charge
- easy to use
- extensible
- One of the biggest developer community
- etc

Outline

Matplotlib : Structure of a program

Matplotlib : Structure of a program

Structure d'un programme

A program with plot is most a the structured in the following fashion :

Matplotlib : Structure of a program

Structure d'un programme

A program with plot is most a the structured in the following fashion :

1. Import the pyplot functionality of matplotlib

```
import matplotlib.pyplot as plt
```

Matplotlib : Structure of a program

Structure d'un programme

A program with plot is most a the structured in the following fashion :

1. Import the pyplot functionality of matplotlib
2. The program part that make the calculations

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))
```

Matplotlib : Structure of a program

Structure d'un programme

A program with plot is most a the structured in the following fashion :

1. Import the pyplot functionality of matplotlib
2. The program part that make the calculations
3. Call of a pyplot function

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))

plt.scatter(X,Y)
```

Matplotlib : Structure of a program

Structure d'un programme

A program with plot is most a the structured in the following fashion :

1. Import the pyplot functionality of matplotlib
2. The program part that make the calculations
3. Call of a pyplot function
4. Ask the plot to be displayed

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))

plt.scatter(X,Y)

plt.show()
```

Outline

Outline

Matplotlib : Scatter points

Scatter points

To plot **Y** as a function of **X**, use the following syntax :

plt.scatter(X,Y)

Matplotlib : Scatter points

Scatter points

To plot **Y** as a function of **X**, use the following syntax :

plt.scatter(X,Y)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))

plt.scatter(X,Y)
plt.show()
```

Matplotlib : Scatter points

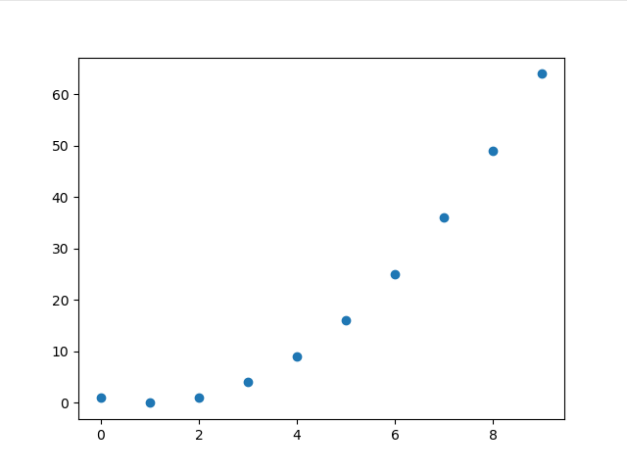
Scatter points

To plot **Y** as a function of **X**, use the following syntax :

plt.scatter(X,Y)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))

plt.scatter(X,Y)
plt.show()
```



Matplotlib : Scatter points

Scatter points : Some options

Change the marker style

Use the option *marker=*

Change the marker size

Use the option *s=* (20 is the default)

Change the marker color

Use the option *c=*

Matplotlib : Scatter points

Scatter points : The various markers style

marker	description
"."	point
",”	pixel
"o"	circle
"v"	triangle_down
"^"	triangle_up
"<"	triangle_left
">"	triangle_right
"1"	tri_down
"2"	tri_up
"3"	tri_left
"4"	tri_right
"8"	octagon
"s"	square
"p"	pentagon
"P"	plus (filled)
"*"	star
"h"	hexagon1
"H"	hexagon2
"+"	plus
"x"	x
"X"	x (filled)
"D"	diamond
"d"	thin_diamond
"|"	vline
"_"	hline

Matplotlib : Scatter points

Scatter points : the various marker colors

value	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Matplotlib : Scatter points

Scatter points : Some options : Example

Matplotlib : Scatter points

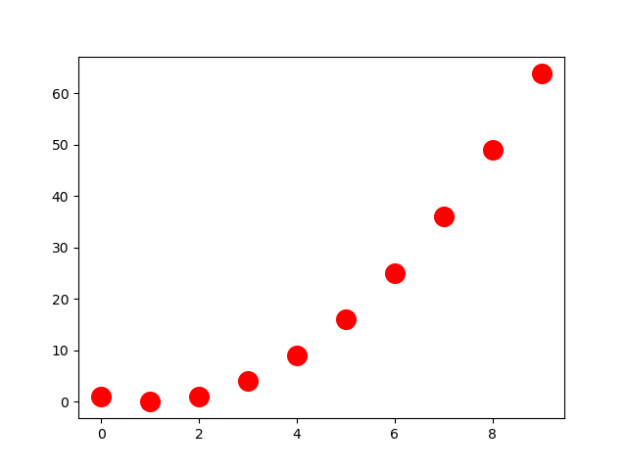
Scatter points : Some options : Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))

plt.scatter(X,Y,marker='o',s=200,c='r')

plt.show()
```

Matplotlib : Scatter points

Scatter points : Some options : Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))

plt.scatter(X,Y,marker='o',s=200,c='r')

plt.show()
```



Matplotlib : Scatter points

Scatter points : Customization

On could also define marker colors and sizes for each points using lists :
(ATTENTION : the size of these lists should be the same as the data lists)

Matplotlib : Scatter points

Scatter points : Customization

On could also define marker colors and sizes for each points using lists :
(ATTENTION : the size of these lists should be the same as the data lists)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
size=[]
c=['r','b','g','k','c','r','b','g','k','c']
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))
        size.append(10*i)
plt.scatter(X,Y,marker='o',s=size,c=c)

plt.show()
```

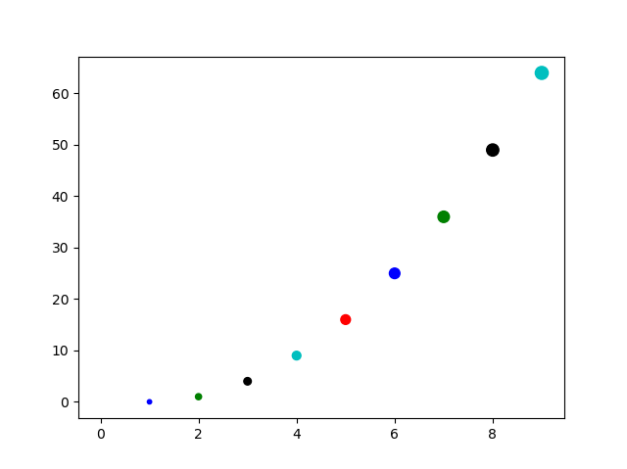
Matplotlib : Scatter points

Scatter points : Customization

On could also define marker colors and sizes for each points using lists :
(ATTENTION : the size of these lists should be the same as the data lists)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
size=[]
c=['r','b','g','k','c','r','b','g','k','c']
for i in range(0,10):
        X.append(i)
        Y.append(calcul(i))
        size.append(10*i)
plt.scatter(X,Y,marker='o',s=size,c=c)

plt.show()
```



Outline

Matplotlib : Curves

Curves

To plot **Y** as a function of **X**, use the following syntax :

plt.plot(X,Y)

Matplotlib : Curves

Curves

To plot **Y** as a function of **X**, use the following syntax :

plt.plot(X,Y)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y)

plt.show()
```

Matplotlib : Curves

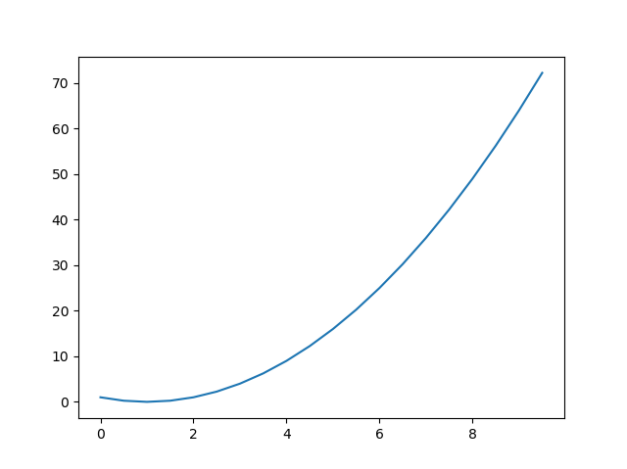
Curves

To plot **Y** as a function of **X**, use the following syntax :

plt.plot(X,Y)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y)

plt.show()
```



Matplotlib : Curves

Matplotlib : Curves

Curves : Some options

Change the linestyle

Use the option *linestyle=*

Change the linewidth

Use the option *linewidth=*

Change the line color

Use the option *color=*

Matplotlib : Curves

Curves : The various line colors

value	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Matplotlib : Curves

Curves : The various line styles

value	linestyle
' :'	doted
'-.'	doted and dashed
'–'	dashed

Matplotlib : Curves

Curves : Some options : Example

Matplotlib : Curves

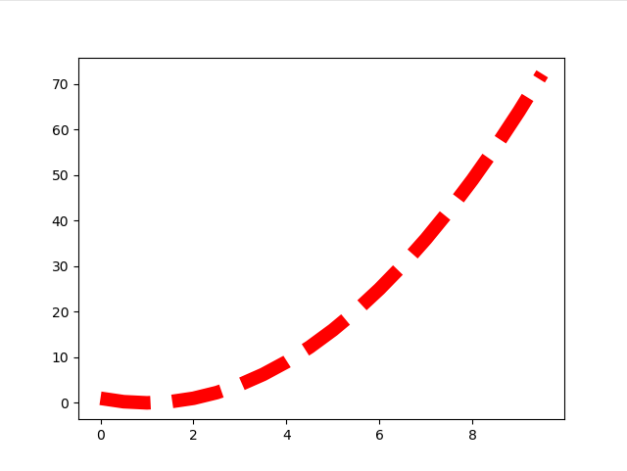
Curves : Some options : Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',linestyle='--', linewidth=10)

plt.show()
```

Matplotlib : Curves

Curves : Some options : Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',linestyle='--', linewidth=10)

plt.show()
```



Outline

Matplotlib : Curves+markers

Curves+markers

It is possible to combine both curves and scatter points.
To do so, use the curves plot method and add the *marker=* option.

Matplotlib : Curves+markers

Curves+markers

It is possible to combine both curves and scatter points.
To do so, use the curves plot method and add the *marker=* option.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o')

plt.show()
```

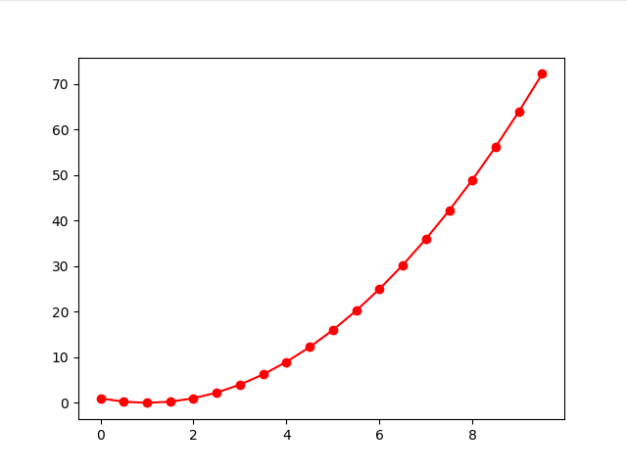
Matplotlib : Curves+markers

Curves+markers

It is possible to combine both curves and scatter points.
To do so, use the curves plot method and add the *marker=* option.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o')

plt.show()
```



Matplotlib : Curves+markers

Matplotlib : Curves+markers

Curves+markers : Some options

Change the marker color

Use the option *markeredgecolor=* and *markerfacecolor*

Change marker size

Use the option *markersize=*

Matplotlib : Curves+markers

Curves+markers : Some options : Example

Matplotlib : Curves+markers

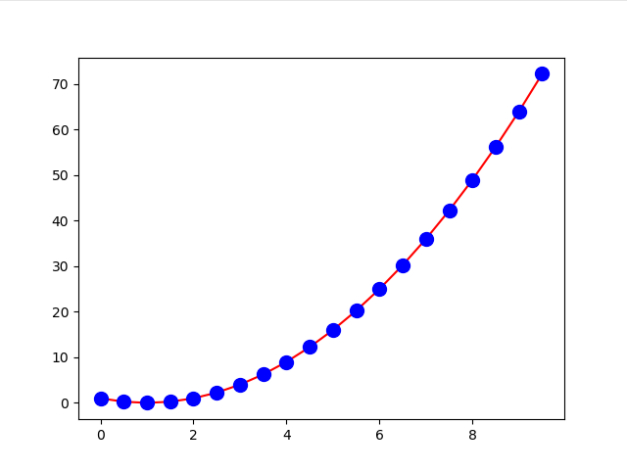
Curves+markers : Some options : Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',
markersize=10,
markerfacecolor='b',markeredgecolor='b')

plt.show()
```

Matplotlib : Curves+markers

Curves+markers : Some options : Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',
markersize=10,
markerfacecolor='b',markeredgecolor='b')

plt.show()
```



Outline

Outline

Matplotlib : Multiples plot

2 plots

If you want to have 2 curves on the same plot, use twice the plot function

Matplotlib : Multiples plot

2 plots

If you want to have 2 curves on the same plot, use twice the plot function

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

plt.plot(X,Y,color='r')
plt.scatter(X,Y1,color='b',marker='o')

plt.show()
```

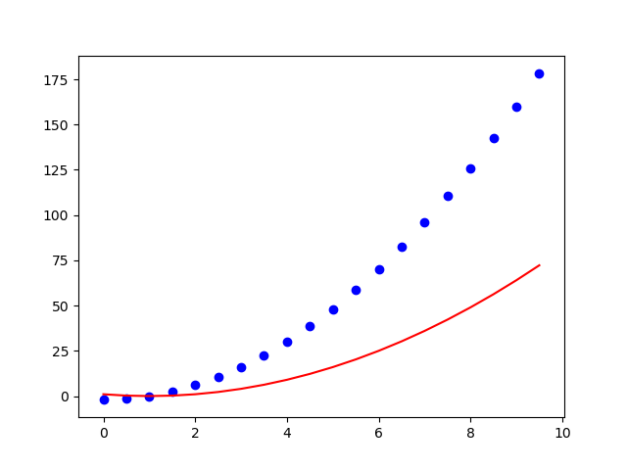
Matplotlib : Multiples plot

2 plots

If you want to have 2 curves on the same plot, use twice the plot function

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

plt.plot(X,Y,color='r')
plt.scatter(X,Y1,color='b',marker='o')

plt.show()
```



Outline

Matplotlib : Multiple plots

2 plots

If you want to have 2 curves on 2 different graphs, use the subplot function

Matplotlib : Multiple plots

2 plots

If you want to have 2 curves on 2 different graphs, use the subplot function

Different type of subplots

- X common axis
- Y common axis
- X and Y common axis
- No common axis

Matplotlib : Multiples plots : X common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2,sharex=True) where *2* specify that you want 2 graphs.

Matplotlib : Multiples plots : X common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2,sharex=True) where *2* specify that you want 2 graphs.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

f, ax = plt.subplots(2, sharex=True)
ax[0].plot(X,Y,color='r')
ax[1].scatter(X,Y1,color='b',marker='o')

plt.show()
```

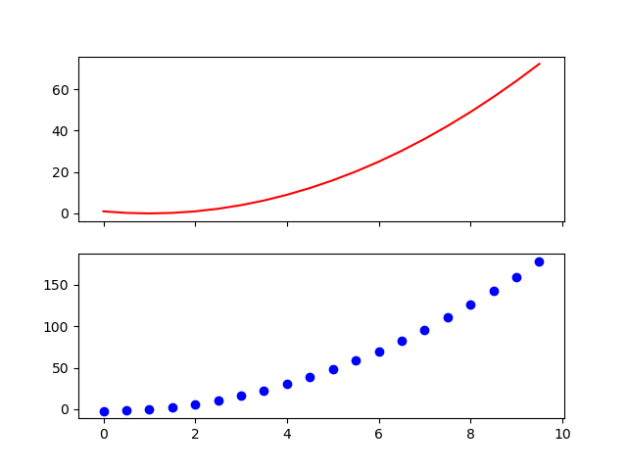
Matplotlib : Multiples plots : X common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2,sharex=True) where *2* specify that you want 2 graphs.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

f, ax = plt.subplots(2, sharex=True)
ax[0].plot(X,Y,color='r')
ax[1].scatter(X,Y1,color='b',marker='o')

plt.show()
```



Matplotlib : Multiples plots : Y common axis

Syntax

Use the subplot function with the syntax :
f, (ax1,ax2) = plt.subplots(1,2,sharex=True) where *1* and *2* specify that you want 1 graph vertically and 2 horizontally.

Matplotlib : Multiples plots : Y common axis

Syntax

Use the subplot function with the syntax :
f, (ax1,ax2) = plt.subplots(1,2,sharex=True) where *1* and *2* specify that you want 1 graph vertically and 2 horizontally.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

f, (ax1,ax2) = plt.subplots(1,2, sharey=True)
ax1.plot(X,Y,color='r')
ax2.scatter(X,Y1,color='b',marker='o')

plt.show()
```

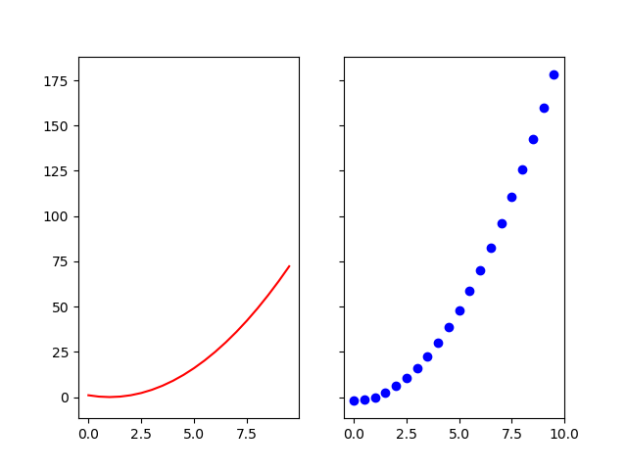
Matplotlib : Multiples plots : Y common axis

Syntax

Use the subplot function with the syntax :
f, (ax1,ax2) = plt.subplots(1,2,sharex=True) where *1* and *2* specify that you want 1 graph vertically and 2 horizontally.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

f, (ax1,ax2) = plt.subplots(1,2, sharey=True)
ax1.plot(X,Y,color='r')
ax2.scatter(X,Y1,color='b',marker='o')

plt.show()
```



Matplotlib : Multiples plots : X and Y common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2,sharex=True, sharey=True)
where *2* specify that you want 2 graphs.

Matplotlib : Multiples plots : X and Y common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2,sharex=True, sharey=True)
where *2* specify that you want 2 graphs.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

f, ax = plt.subplots(2, sharey=True, sharex=True)
ax[0].plot(X,Y,color='r')
ax[1].scatter(X,Y1,color='b',marker='o')

plt.show()
```

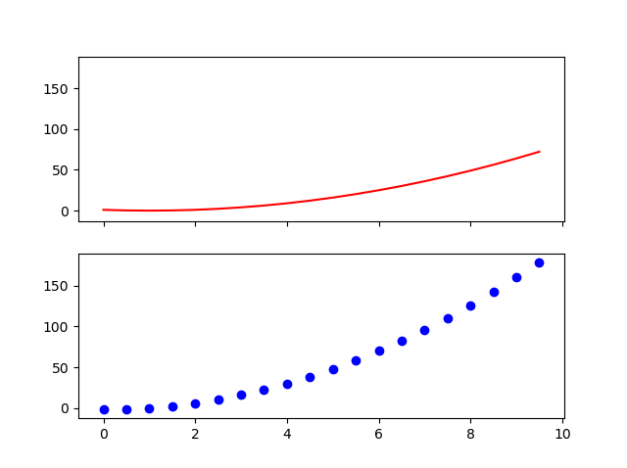
Matplotlib : Multiples plots : X and Y common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2,sharex=True, sharey=True)
where *2* specify that you want 2 graphs.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i/2.)[1])

f, ax = plt.subplots(2, sharey=True, sharex=True)
ax[0].plot(X,Y,color='r')
ax[1].scatter(X,Y1,color='b',marker='o')

plt.show()
```



Matplotlib : Multiples plots : No common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2) where *2* specify that you want 2 graphs.

Matplotlib : Multiples plots : No common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2) where *2* specify that you want 2 graphs.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, ax = plt.subplots(2)
ax[0].plot(X,Y,color='r')
ax[1].scatter(X1,Y1,color='b',marker='o')

plt.show()
```

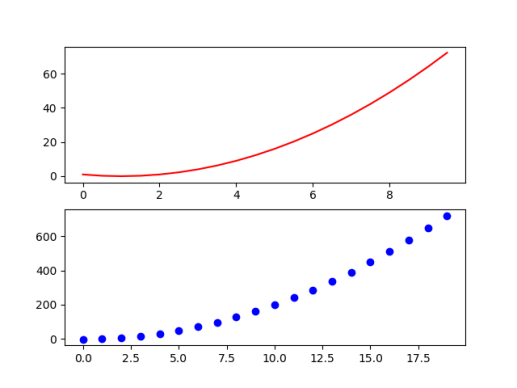
Matplotlib : Multiples plots : No common axis

Syntax

Use the subplot function with the syntax :
f, ax = plt.subplots(2) where *2* specify that you want 2 graphs.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, ax = plt.subplots(2)
ax[0].plot(X,Y,color='r')
ax[1].scatter(X1,Y1,color='b',marker='o')

plt.show()
```



Outline

Outline

Graph title : single graph

Add a title to a graph

To add a graph title, use the following syntax :
plt.title(title_name)

Graph title : single graph

Add a title to a graph

To add a graph title, use the following syntax :
plt.title(title_name)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o')
plt.title(r'$X^2-2X+1$')
plt.show()
```

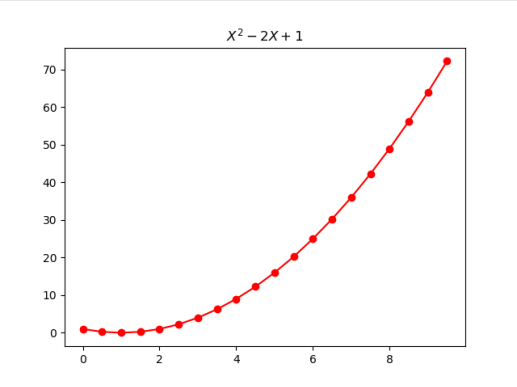
Graph title : single graph

Add a title to a graph

To add a graph title, use the following syntax :
plt.title(title_name)

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o')
plt.title(r'$X^2-2X+1$')
plt.show()
```



Graph title : Subplots

Add a subplot title

To a dd a global graph title, use :
suptitle(title_name)
To add a subplot title, use :
set_title(title_name)

Graph title : Subplots

Add a subplot title

To a dd a global graph title, use :
suptitle(title_name)
To add a subplot title, use :
set_title(title_name)

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r')
ax2.scatter(X1,Y1,color='b',marker='o')

ax1.set_title('Graph1')
ax2.set_title('Graph2')

plt.show()
```

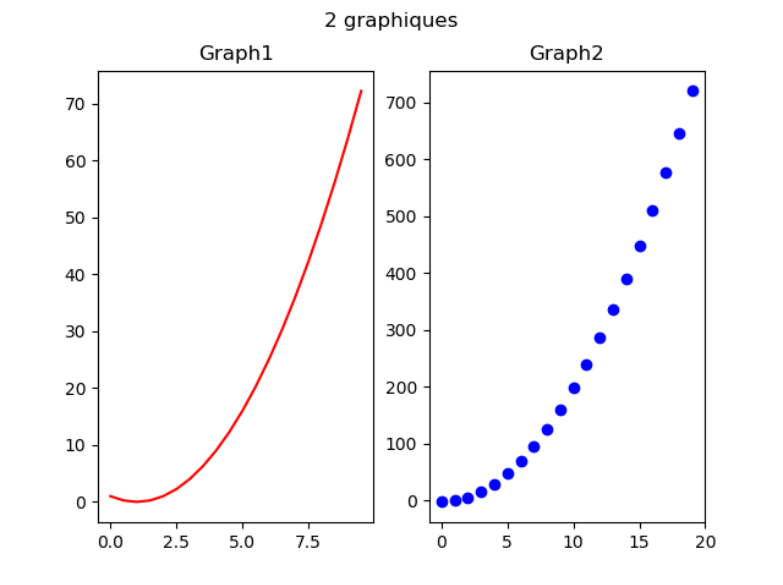
Graph title : Subplots

Add a subplot title

To a dd a global graph title, use :
suptitle(title_name)
To add a subplot title, use :
set_title(title_name)

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r')
ax2.scatter(X1,Y1,color='b',marker='o')

ax1.set_title('Graph1')
ax2.set_title('Graph2')

plt.show()
```



Outline

Axis title : Single graph

Add axix titles

To add axis titles, use :
plt.xlabel(X_axis_title) and *plt.ylabel(Y_axis_title)*

Axis title : Single graph

Add axix titles

To add axis titles, use :
plt.xlabel(X_axis_title) and *plt.ylabel(Y_axis_title)*

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

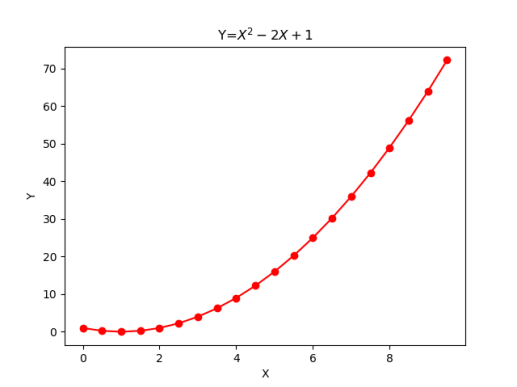
Axis title : Single graph

Add axix titles

To add axis titles, use :
plt.xlabel(X_axis_title) and *plt.ylabel(Y_axis_title)*

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



Axis titles : Subplots

Add axis titles

To add axis titles, use :
set_xlabel(X_axis_title) et *set_ylabel(Y_axis_title)*

Axis titles : Subplots

Add axis titles

To add axis titles, use :
set_xlabel(X_axis_title) et *set_ylabel(Y_axis_title)*

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r')
ax2.scatter(X1,Y1,color='b',marker='o')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('Z')
ax1.set_ylabel(r'Y=$X^2-2X+1$')
ax2.set_ylabel(r'Y=$2X^2-4X+2$')
plt.tight_layout()
plt.show()
```

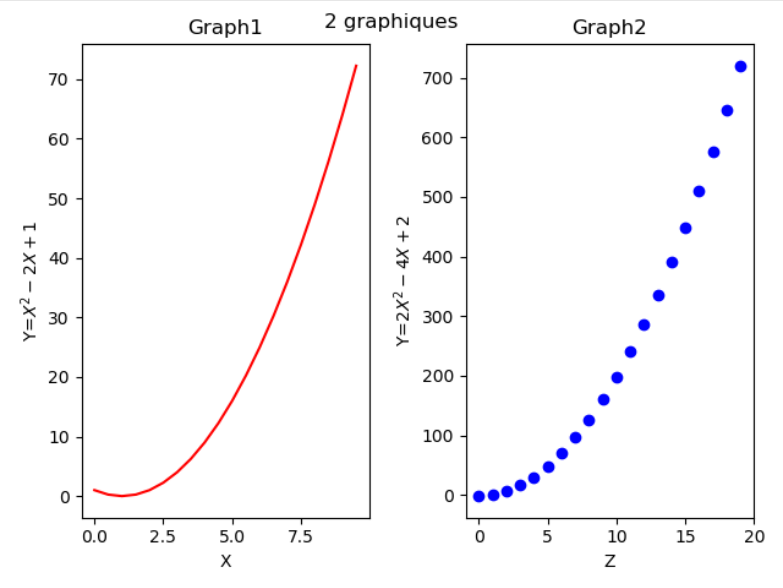
Axis titles : Subplots

Add axis titles

To add axis titles, use :
set_xlabel(X_axis_title) et *set_ylabel(Y_axis_title)*

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r')
ax2.scatter(X1,Y1,color='b',marker='o')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('Z')
ax1.set_ylabel(r'Y=$X^2-2X+1$')
ax2.set_ylabel(r'Y=$2X^2-4X+2$')
plt.tight_layout()
plt.show()
```



Outline

Legend : single graphique

Add a legend

To add a legend, use :
label= option for each plot and *plt.legend* to show legend.

Legend : single graphique

Add a legend

To add a legend, use :
label= option for each plot and *plt.legend* to show legend.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',label=r'Y=$X^2-2X+1$')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

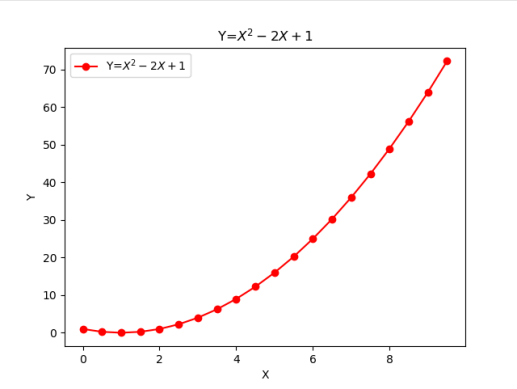
Legend : single graphique

Add a legend

To add a legend, use :
label= option for each plot and *plt.legend* to show legend.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',label=r'Y=$X^2-2X+1$')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



legend : Subplots

Add a legend

PTo add a legend to each subplots, use :
label= option for each plot and *.legend* to show legends.

legend : Subplots

Add a legend

PTo add a legend to each subplots, use : *label=* option for each plot and *.legend* to show legends.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r',label=r'Y=$X^2-2X+1$')
ax2.scatter(X1,Y1,color='b',marker='o',
label=r'Y=$2X^2-4X+2$')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('X')
ax1.legend()
ax2.legend()
ax2.set_ylabel('Y')
ax1.set_ylabel('Y')
plt.tight_layout()
plt.show()
```

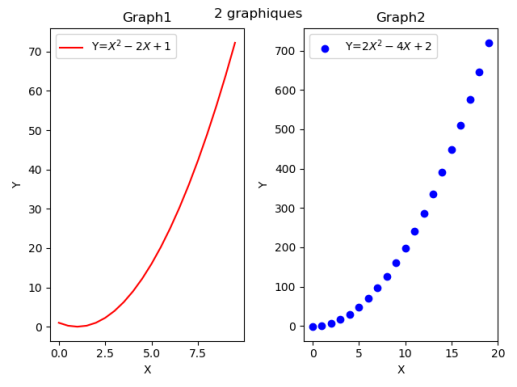
legend : Subplots

Add a legend

PTo add a legend to each subplots, use : *label=* option for each plot and *.legend* to show legends.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r',label=r'Y=$X^2-2X+1$')
ax2.scatter(X1,Y1,color='b',marker='o',
label=r'Y=$2X^2-4X+2$')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('X')
ax1.legend()
ax2.legend()
ax2.set_ylabel('Y')
ax1.set_ylabel('Y')
plt.tight_layout()
plt.show()
```



Outline

Axis range : Single graphique

Define X and Y range

To define X and Y ranges, use :
xlim=(min,max) and *ylim=(min,max)*.

Axis range : Single graphique

Define X and Y range

To define X and Y ranges, use :
xlim=(min,max) and *ylim=(min,max)*.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',label=r'Y=$X^2-2X+1$')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

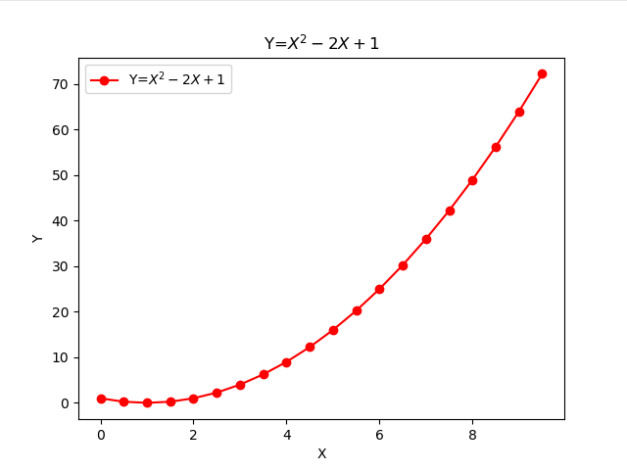
Axis range : Single graphique

Define X and Y range

To define X and Y ranges, use :
xlim=(min,max) and *ylim=(min,max)*.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',label=r'Y=$X^2-2X+1$')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



Axis range : Single graphique

Define X and Y range

To define X and Y ranges, use :
xlim=(min,max) and *ylim=(min,max)*.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',label=r'Y=$X^2-2X+1$')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.xlim(0.0,10.0)
plt.ylim(0.0,80.0)
plt.show()
```

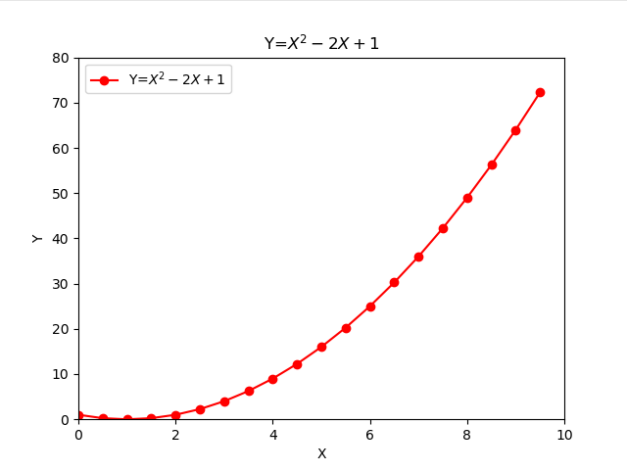
Axis range : Single graphique

Define X and Y range

To define X and Y ranges, use :
xlim=(min,max) and *ylim=(min,max)*.

Example

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        return(B)

X=[]
Y=[]
for i in range(0,20):
        X.append(i/2.)
        Y.append(calcul(i/2.))

plt.plot(X,Y,color='r',marker='o',label=r'Y=$X^2-2X+1$')
plt.title(r'Y=$X^2-2X+1$')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.xlim(0.0,10.0)
plt.ylim(0.0,80.0)
plt.show()
```



Axis range : Subplots

Define X and Y range

PTo define X and Y ranges, use : *set_xlim=(min,max)* and *set_ylim=(min,max)* for each subplots.

Axis range : Subplots

Define X and Y range

PTo define X and Y ranges, use : *set_xlim=(min,max)* and *set_ylim=(min,max)* for each subplots.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r',label=r'Y=$X^2-2X+1$')
ax2.scatter(X1,Y1,color='b',marker='o',
label=r'Y=$2X^2-4X+2$')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('X')
ax1.legend()
ax2.legend()
ax2.set_ylabel('Y')
ax1.set_ylabel('Y')
plt.tight_layout()
plt.show()
```

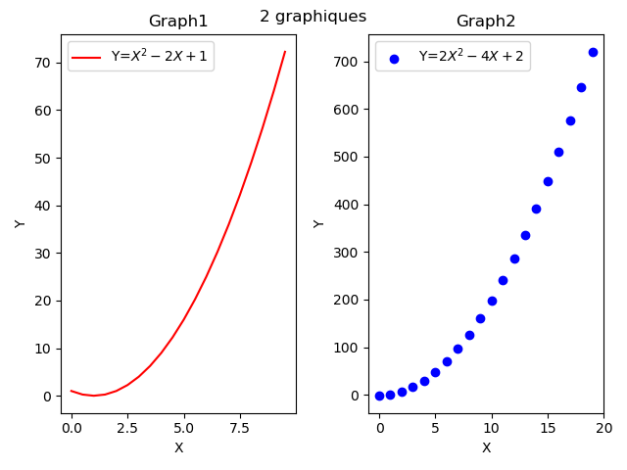
Axis range : Subplots

Define X and Y range

PTo define X and Y ranges, use : *set_xlim=(min,max)* and *set_ylim=(min,max)* for each subplots.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r',label=r'Y=$X^2-2X+1$')
ax2.scatter(X1,Y1,color='b',marker='o',
label=r'Y=$2X^2-4X+2$')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('X')
ax1.legend()
ax2.legend()
ax2.set_ylabel('Y')
ax1.set_ylabel('Y')
plt.tight_layout()
plt.show()
```



Axis range : Subplots

Define X and Y range

PTo define X and Y ranges, use : *set_xlim=(min,max)* and *set_ylim=(min,max)* for each subplots.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r',label=r'Y=$X^2-2X+1$')
ax2.scatter(X1,Y1,color='b',marker='o',
label=r'Y=$2X^2-4X+2$')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('X')
ax1.legend()
ax2.legend()
ax2.set_ylabel('Y')
ax1.set_ylabel('Y')
ax1.set_xlim(0,10)
ax1.set_ylim(0,80)
ax2.set_xlim(0,20)
ax2.set_ylim(0,800)
plt.tight_layout()
plt.show()
```

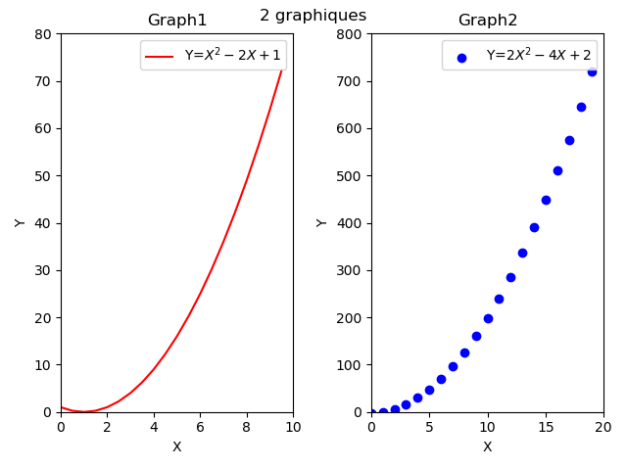
Axis range : Subplots

Define X and Y range

PTo define X and Y ranges, use : *set_xlim=(min,max)* and *set_ylim=(min,max)* for each subplots.

Exemple

```
import matplotlib.pyplot as plt

def calcul(A):
        B=A**2-2*A+1
        C=2*A**2-4*1+2
        return(B,C)

X=[]
X1=[]
Y=[]
Y1=[]
for i in range(0,20):
        X.append(i/2.)
        X1.append(i)
        Y.append(calcul(i/2.)[0])
        Y1.append(calcul(i)[1])

f, [ax1,ax2] = plt.subplots(1,2)
f.suptitle('2 graphiques')
ax1.plot(X,Y,color='r',label=r'Y=$X^2-2X+1$')
ax2.scatter(X1,Y1,color='b',marker='o',
label=r'Y=$2X^2-4X+2$')

ax1.set_title('Graph1')
ax2.set_title('Graph2')
ax1.set_xlabel('X')
ax2.set_xlabel('X')
ax1.legend()
ax2.legend()
ax2.set_ylabel('Y')
ax1.set_ylabel('Y')
ax1.set_xlim(0,10)
ax1.set_ylim(0,80)
ax2.set_xlim(0,20)
ax2.set_ylim(0,800)
plt.tight_layout()
plt.show()
```



3D plots with matplotlib

Outline

Matplotlib3D : Import modules

Matplotlib3D : Import modules

As for 2D

As for 2D graphs, you should import pyplot :
import matplotlib.pyplot as plt

Matplotlib3D : Import modules

As for 2D

As for 2D graphs, you should import pyplot :
import matplotlib.pyplot as plt

Addition module for 3D

For 3D plots, you should also import the mplot3D function of the mpl_toolkits module :
from mpl_toolkits.mplot3d import Axes3D

Matplotlib3D : Specific syntax

Matplotlib3D : Specific syntax

Create a figure

You should create a particular interface for the figure specifying that it will be a 3D figure :

fig = plt.figure()
ax = fig.gca(projection='3d')

Outline

Matplotlib3D : Curves

plot 3D curves

It works exactly as for 2D except that you should give 3 data lists :
plt.plot(X,Y,Z)

Matplotlib3D : Curves

plot 3D curves

It works exactly as for 2D except that you should give 3 data lists :
plt.plot(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
x=[]
y=[]
z=[]
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
for i in range(0,30):
        for j in range(0,30):
                x.append(np.cos(u[i])*np.cos(v[j]))
                y.append(np.sin(u[i])*np.cos(v[j]))
                z.append(np.sin(v[j]))
ax.plot(x, y, z, label='une sphere')
ax.legend()

plt.show()
```

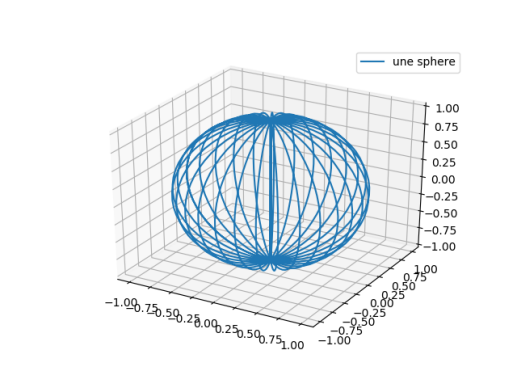
Matplotlib3D : Curves

plot 3D curves

It works exactly as for 2D except that you should give 3 data lists :
plt.plot(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
x=[]
y=[]
z=[]
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
for i in range(0,30):
        for j in range(0,30):
                x.append(np.cos(u[i])*np.cos(v[j]))
                y.append(np.sin(u[i])*np.cos(v[j]))
                z.append(np.sin(v[j]))
ax.plot(x, y, z, label='une sphere')
ax.legend()

plt.show()
```



Outline

Matplotlib3D : Scatter points

3D Scatter points

It works exactly as for 2D except that you should give 3 data lists :

plt.scatter(X,Y,Z)

Matplotlib3D : Scatter points

3D Scatter points

It works exactly as for 2D except that you should give 3 data lists :

plt.scatter(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
x=[]
y=[]
z=[]
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
for i in range(0,30):
        for j in range(0,30):
                x.append(np.cos(u[i])*np.cos(v[j]))
                y.append(np.sin(u[i])*np.cos(v[j]))
                z.append(np.sin(v[j]))
ax.scatter(x, y, z, s=5, label='une sphere')
ax.legend()

plt.show()
```

Matplotlib3D : Scatter points

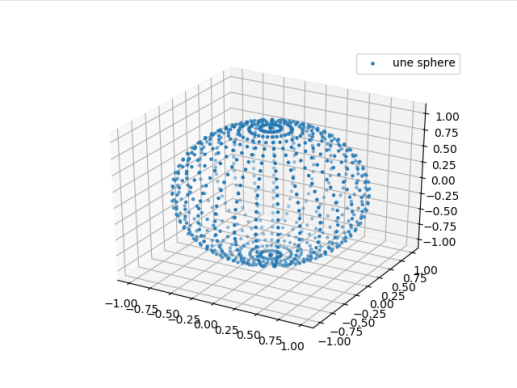
3D Scatter points

It works exactly as for 2D except that you should give 3 data lists :

plt.scatter(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
x=[]
y=[]
z=[]
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
for i in range(0,30):
        for j in range(0,30):
                x.append(np.cos(u[i])*np.cos(v[j]))
                y.append(np.sin(u[i])*np.cos(v[j]))
                z.append(np.sin(v[j]))
ax.scatter(x, y, z, s=5, label='une sphere')
ax.legend()

plt.show()
```



Outline

Matplotlib3D : Wireframe

Wireframe plots

To do this, use :
plt.wireframe(X,Y,Z)

Matplotlib3D : Wireframe

Wireframe plots

To do this, use :

plt.wireframe(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
x=np.zeros((30,30))
y=np.zeros((30,30))
z=np.zeros((30,30))
for i in range(0,30):
        for j in range(0,30):
                x[i][j]=np.cos(u[i])*np.cos(v[j])
                y[i][j]=np.sin(u[i])*np.cos(v[j])
                z[i][j]=np.sin(v[j])
ax.plot_wireframe(x,y,z, label='une sphere')
ax.legend()

plt.show()
```

Matplotlib3D : Wireframe

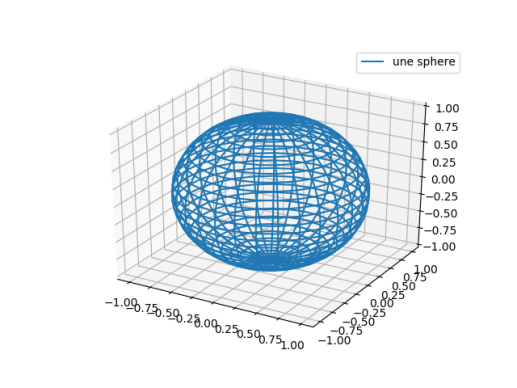
Wireframe plots

To do this, use :

plt.wireframe(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
x=np.zeros((30,30))
y=np.zeros((30,30))
z=np.zeros((30,30))
for i in range(0,30):
        for j in range(0,30):
                x[i][j]=np.cos(u[i])*np.cos(v[j])
                y[i][j]=np.sin(u[i])*np.cos(v[j])
                z[i][j]=np.sin(v[j])
ax.plot_wireframe(x,y,z, label='une sphere')
ax.legend()

plt.show()
```



Outline

Matplotlib3D : Surface

Surface plot

You should use the following syntax :
plt.surface(X,Y,Z)

Matplotlib3D : Surface

Surface plot

You should use the following syntax :
plt.surface(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
x=np.zeros((30,30))
y=np.zeros((30,30))
z=np.zeros((30,30))
for i in range(0,30):
        for j in range(0,30):
                x[i][j]=np.cos(u[i])*np.cos(v[j])
                y[i][j]=np.sin(u[i])*np.cos(v[j])
                z[i][j]=np.sin(v[j])
ax.plot_surface(x,y,z, color='b')

plt.show()
```

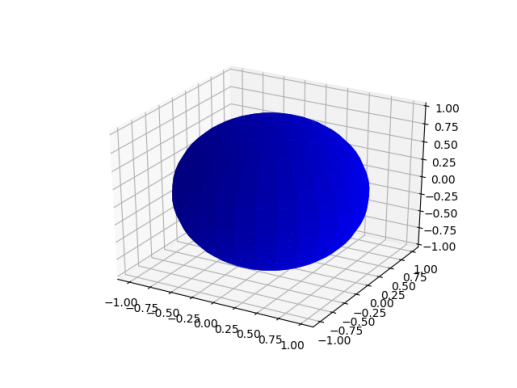
Matplotlib3D : Surface

Surface plot

You should use the following syntax :
plt.surface(X,Y,Z)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
x=np.zeros((30,30))
y=np.zeros((30,30))
z=np.zeros((30,30))
for i in range(0,30):
        for j in range(0,30):
                x[i][j]=np.cos(u[i])*np.cos(v[j])
                y[i][j]=np.sin(u[i])*np.cos(v[j])
                z[i][j]=np.sin(v[j])
ax.plot_surface(x,y,z, color='b')

plt.show()
```



Matplotlib3D : Surface plot

With a color gradient

You can do it using the cmap=cm.colorwarm option :
plt.surface(X,Y,Z, cmap=cm.colorwarm)

Matplotlib3D : Surface plot

With a color gradient

You can do it using the cmap=cm.colorwarm option :
plt.surface(X,Y,Z, cmap=cm.colorwarm)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
x=np.zeros((30,30))
y=np.zeros((30,30))
z=np.zeros((30,30))
for i in range(0,30):
        for j in range(0,30):
                x[i][j]=np.cos(u[i])*np.cos(v[j])
                y[i][j]=np.sin(u[i])*np.cos(v[j])
                z[i][j]=np.sin(v[j])
surf=ax.plot_surface(x,y,z, cmap=cm.coolwarm)
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

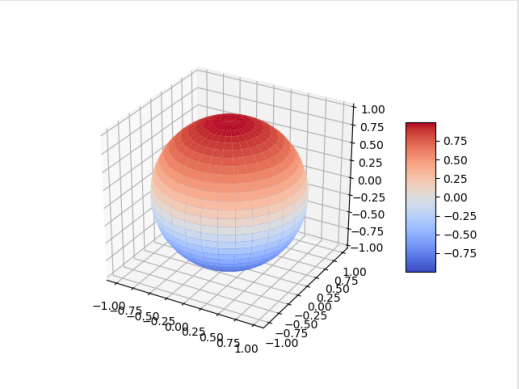
Matplotlib3D : Surface plot

With a color gradient

You can do it using the cmap=cm.colorwarm option :
plt.surface(X,Y,Z, cmap=cm.colorwarm)

Example

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

fig = plt.figure()
ax = fig.gca(projection='3d')
u=np.linspace(-1*np.pi,np.pi,30)
v=np.linspace(-1*np.pi/2,np.pi/2,30)
x=np.zeros((30,30))
y=np.zeros((30,30))
z=np.zeros((30,30))
for i in range(0,30):
        for j in range(0,30):
                x[i][j]=np.cos(u[i])*np.cos(v[j])
                y[i][j]=np.sin(u[i])*np.cos(v[j])
                z[i][j]=np.sin(v[j])
surf=ax.plot_surface(x,y,z, cmap=cm.coolwarm)
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```



Outline

Matplotlib3D : 2D plots

Plot 2D

This can be done by removing a direction (z for example) :
ax.plot(x, y, zs=0, zdir='z')

Matplotlib3D : 2D plots

Plot 2D

This can be done by removing a direction (z for example) :
ax.plot(x, y, zs=0, zdir='z')

Exemple

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

fig = plt.figure()
ax = fig.gca(projection='3d')
x = np.linspace(0, 1, 100)
y = np.sin(x * 2 * np.pi) / 2 + 0.5
ax.plot(x, y, zs=0, zdir='z', label='curve in (x,y)')
colors = ['r', 'g', 'b', 'k']
X1 = np.random.sample(20*len(colors))
Y1 = np.random.sample(20*len(colors))
c_list = []
for i in range(0,20):
        for c in range(0,len(colors)):
                c_list.append(colors[c])
ax.scatter(X1, Y1, zs=0, zdir='y', c=c_list)
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_zlim(0, 1)
plt.show()
```

Matplotlib3D : 2D plots

Plot 2D

This can be done by removing a direction (z for example) :
ax.plot(x, y, zs=0, zdir='z')

Exemple

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

fig = plt.figure()
ax = fig.gca(projection='3d')
x = np.linspace(0, 1, 100)
y = np.sin(x * 2 * np.pi) / 2 + 0.5
ax.plot(x, y, zs=0, zdir='z', label='curve in (x,y)')
colors = ['r', 'g', 'b', 'k']
X1 = np.random.sample(20*len(colors))
Y1 = np.random.sample(20*len(colors))
c_list = []
for i in range(0,20):
        for c in range(0,len(colors)):
                c_list.append(colors[c])
ax.scatter(X1, Y1, zs=0, zdir='y', c=c_list)
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_zlim(0, 1)
plt.show()
```