

Programmation en Python



Introduction

La distribution Anaconda

Présentation de Jupyter

Outils Logiciels pour l'enseignement supérieur

Les librairies scientifiques

Programmation de base :

- Déclaration de variables

- Les opérateurs

- Les boucles

- Les fonctions

- Les courbes

- Les nombres complexes

Python est un langage de programmation interprété, c'est-à-dire que les instructions que vous lui envoyez sont "transcrites" en langage machine au fur et à mesure de leur lecture.

Les avantages d'un langage interprété sont la simplicité (on ne passe pas par une étape de compilation avant d'exécuter son programme) et la portabilité (un langage tel que Python est censé fonctionner aussi bien sous Windows que sous Linux ou Mac OS, et on ne devrait avoir à effectuer aucun changement dans le code pour le passer d'un système à l'autre). Cela ne veut pas dire que les langages compilés ne sont pas portables, loin de là ! Mais on doit utiliser des compilateurs différents et, d'un système à l'autre, certaines instructions ne sont pas compatibles, voire se comportent différemment.

En contrepartie, un langage compilé se révélera bien plus rapide qu'un langage interprété (la traduction à la volée de votre programme ralentit l'exécution), bien que cette différence tende à se faire de moins en moins sentir au fil des améliorations.

Présentation de la distribution Anaconda

- ▶ Distribution Python 3 dédié aux calculs scientifiques.
- ▶ Environnement de Développement Intégré (Spyder)
- ▶ Outils de création d'interface (Qt Designer,...)
- ▶ Large panel de fonctionnalités scientifiques

...	
Matplotlib	Scipy
Numpy	
Python 3	

Figure 1: Distribution Scientifique

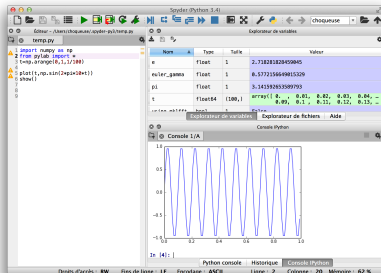


Figure 2: Environnement Spyder


La distribution Anaconda - Ouverture d'Anaconda

Page d'accueil Anaconda



[Home](#)
[Environments](#)
[Learning](#)
[Community](#)
[Documentation](#)
[Developer Blog](#)


Applications on base (root) Channels



JupyterLab
1.2.6

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.


[Launch](#)



Jupyter Notebook
6.0.3

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.


[Launch](#)



Qt Console
4.6.0

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.


[Launch](#)



Spyder
4.0.1

Scientific Python Development Environment; Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features


[Launch](#)



VS Code
1.63.2

Streamlined code editor with support for development operations like debugging, task running and version control.


[Launch](#)



Glueviz
0.15.2

Multidimensional data visualization across files. Explore relationships within and among related datasets.


[Install](#)



Orange 3
3.23.1

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

[Install](#)



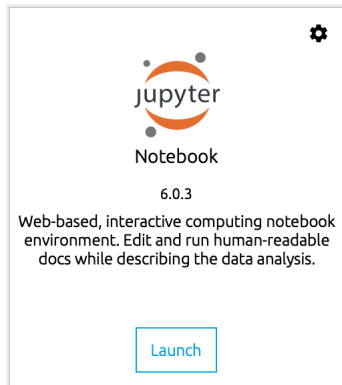
RStudio
1.1.456

A set of integrated tools designed to help you be more productive with R includes R essentials and notebooks.

[Install](#)

La distribution Anaconda - Ouverture de Jupyter

Ouvrir Jupyter Notebook en cliquant sur "Launch"



2ème méthodes

- Dans un terminal, lancer la ligne de commande :

```
$ jupyter notebook
```

Jupyter Notebook

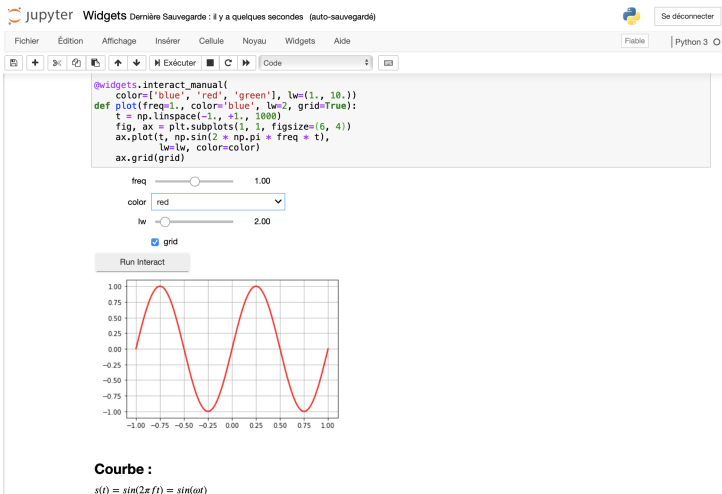


Figure 3: Interface d'un notebook jupyter

Interface des notebooks

- ▶ Tout se passe dans le navigateur web (Firefox, Edge, Safari, etc)
- ▶ Un notebook est composé de "cellules"
- ▶ Une cellule peut contenir du
 - ▶ code python,
 - ▶ du code markdown (edition du texte).

The screenshot shows a Jupyter Notebook interface with several cells. Annotations with arrows point to specific features:

- Cellule Markdown avec Latex:** Points to a cell containing text and a LaTeX equation $X(f) = \int_{-\infty}^{\infty} x(t)e^{-2\pi ift} dt$.
- Choix du type de cellule:** Points to the dropdown menu in the top right of the cell toolbar, which is currently set to 'Markdown'.
- Cellule Markdown:** Points to a cell titled 'Exercice 1:' containing text.
- Cellules Python:** Points to a code cell containing Python code for plotting a signal.
- Résultat après execution:** Points to the output area of the code cell, which displays a plot of a signal with several pulses.

Figure 4: Les différents types de cellules.

Le Markdown

- ▶ En bref
 - ▶ Langage simple plus concis que le Latex ou le HTML.
 - ▶ Possibilités de conversions multiples via l'utilitaire pandoc.
- ▶ Liste des commandes : <https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

Exemple:

```
# Ceci est un titre  
## Ceci est un sous titre  
  
Ce qui suit est une liste classique  
  
* ceci est un item  
* ceci est également un item  
  
Nous pouvons également faire des  
listes numerotees  
  
1. ceci est un item numerote  
2. idem
```

Ceci est un titre

Ceci est un sous titre

Ce qui suit est une liste classique

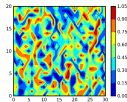
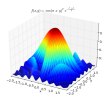
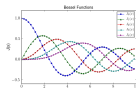
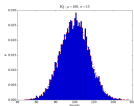
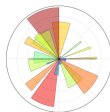
- ceci est un item
- ceci est également un item

Nous pouvons également faire des listes numérotées

1. ceci est un item numéroté
2. idem

Cahier des charges

- ▶ Fonctions mathématiques poussées (fonctions trigonométriques, complexes, ...etc)
- ▶ Outils de visualisation 2D, 3D,
- ▶ Outils de calcul formel,
- ▶ ...



Quelques outils possibles

Matlab, Scilab, ... Python !

En bref

- ▶ Créateur: Guido van Rossum.
- ▶ Versions:
 - ▶ Python 1 (1990)
 - ▶ ...
 - ▶ **Python 3.13 (07 octobre 2024)**
- ▶ Caractéristiques du langage:
 - ▶ Multiplateforme.
 - ▶ Langage de script.
 - ▶ Programmation
procédurale, objet,...



Python par rapport aux autres langages

Classement TIOBE (Août 2024)

Aug 2024	Aug 2023	Change	Programming Language	Ratings	Change
1	1		 Python	18.04%	+4.71%
2	3	▲	 C++	10.04%	-0.59%
3	2	▼	 C	9.17%	-2.24%
4	4		 Java	9.16%	-1.16%
5	5		 C#	6.39%	-0.65%
6	6		 JavaScript	3.91%	+0.62%
7	8	▲	 SQL	2.21%	+0.68%
8	7	▼	 Visual Basic	2.18%	-0.45%
9	12	▲	 Go	2.03%	+0.87%
10	14	▲	 Fortran	1.79%	+0.75%
11	13	▲	 MATLAB	1.72%	+0.67%
12	23	▲	 Delphi/Object Pascal	1.63%	+0.83%
13	10	▼	 PHP	1.46%	+0.19%

Exemple

Langage C

```
#include <stdio.h>

int main( void )
{
    long long int k, S=0;
    for (k=0; k<10000000; k++)
    {
        S=S+k;
    }
    printf ("%lli\n", S);
    return 0;
}
```

Python

```
S=0
for k in range(10000000):
    S=S+k
print (S)
```

Figure 5: Exemple: Somme d'une suite arithmétique ($S = 0 + 1 + \dots + N$)

- ▶ Python est un langage plus concis et plus haut niveau que le langage C. Il permet de **coder des applications plus rapidement**.
- ▶ C reste un langage nettement plus rapide à l'exécution.

C'est un regroupement de fonctions prédéfinies, qui une fois importées permettent d'accéder à de nouvelles fonctions.

Ces bibliothèques (modules ou bibliothèques) sont importées au début du programme via l'instruction `import`.

On peut citer :

- ▶ `numpy` : permet de faire du calcul scientifique
- ▶ `scipy` : permet de faire du traitement du signal ...
- ▶ `matplotlib` : permet de faire des graphiques
- ▶ `statistics` : permet de calculer des valeurs statistiques basiques (moyenne, médiane, variance, ...)
- ▶ ...

- ▶ `import module` : il faudra alors faire précéder les fonctions de ce module du préfixe module. Par exemple `math.sqrt(2)` va renvoyer la racine carrée de 2 avec le module `math` si on a écrit en début de programme : `import math`
- ▶ `from module import fonction` : la fonction "`fonction`" peut alors s'utiliser directement (recommandé). Par exemple `sqrt(2)` va renvoyer la racine carrée de 2 avec le module `math` si on a écrit en début de programme : `from math import sqrt`
- ▶ `from module import *` : toutes les fonctions du module "`module`" sont alors importées (à éviter). Par exemple `sqrt(2)` va renvoyer la racine carrée de 2 avec le module `math` si on a écrit en début de programme : `from math import *`
- ▶ `import module as md` : toutes les fonctions du module "`module`" sont alors importées (à privilégier). Par exemple `np.sqrt(2)` va renvoyer la racine carrée de 2 avec le module `numpy` si on a écrit en début de programme : `import numpy as np`

► Instruction inconnue

```
cos(pi)
```

```
NameError                                Traceback (  
    most recent call last)  
<ipython-input-4-a971a9dfc2c9> in <module>  
----> 1 cos(pi)
```

```
NameError: name 'cos' is not defined
```

► Solution 1

```
from math import *  
cos(pi)
```

```
-1.0
```

► Solution 2

```
import numpy as np  
np.cos(np.pi)
```

```
-1.0
```


Numpy (from numpy import *)

Librairie permettant la manipulation et les opérations sur les tableaux, vecteurs, et matrices, etc.

Code Listing 1: Sans Numpy

```
from math import *  
  
angles=[0, pi /6, pi /4, pi /3, pi /2]  
  
mes_cosinus=[]  
  
for k in range(len(angles)):  
    valeur=cos(angles[k])  
    mes_cosinus.append(valeur)  
  
print(mes_cosinus)
```

Code Listing 2: Avec Numpy

```
from numpy import *  
  
angles=[0, pi /6, pi /4, pi /3, pi /2]  
  
#creation d'un tableau numpy  
tableau=array(angles)  
  
mes_cosinus=cos(tableau)  
  
print(mes_cosinus)
```

Figure 6: Exemple: Calcul des cosinus de plusieurs angles

Matplotlib (from matplotlib.pyplot import *)

Librairie permettant l'affichage et l'export de courbes (2D, 3D).

```
from numpy import *
from matplotlib.pyplot import *

f0=2
#creation de la base temps
t=arange(1000)/1000

#creation de la sinusoïde
x=cos(2*pi*f0*t)

#affichage
plot(t,x)
xlabel('temps(s)')
ylabel('sinusoïde')
show()
```

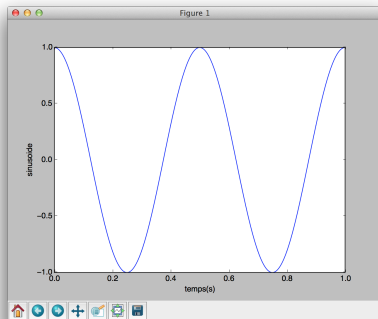


Figure 7: Exemple: Affichage de la sinusoïde $x(t) = \cos(2\pi f_0 t)$ avec $f_0 = 2$ Hz et $t = \frac{n}{1000}$ ($n \in \mathbb{N}$)

Scipy (from scipy import *)

```
from scipy.io import *
from matplotlib.pyplot import *
from skimage import color
from skimage import filters

M1=imread('img/lenna.png')
M2=color.rgb2gray(M1)
val = filters.threshold_otsu(M2)
M3 = M2 > val
M4=filters.gaussian_filter(M2, sigma=5)

f, ax= subplots(2, 2)
ax[0,0].imshow(M1)
ax[0,1].imshow(M2,cmap=cm.gray)
ax[1,0].imshow(M3,cmap=cm.gray)
ax[1,1].imshow(M4,cmap=cm.gray)
show()
```

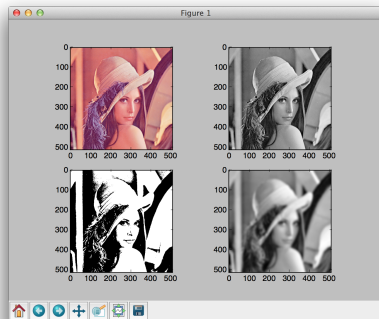


Figure 8: Exemple: Traitement d'images

...Pour aller plus loin

Conception d'Interface Utilisateur (PyQt, TkInter, Kivy,...)

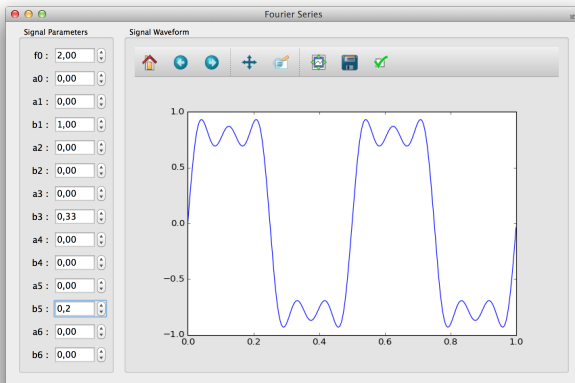


Figure 9: Exemple: Interface réalisée avec PyQt5

Développement de sites Web

- ▶ Exemples de sites utilisant le framework Python/Django
 - ▶ Instagram,
 - ▶ Spotify,
 - ▶ Prezi,
 - ▶ ...
- ▶ Exemples de sites utilisant d'autres solutions basées sur Python
 - ▶ Youtube,
 - ▶ Dropbox,
 - ▶ ...
 - ▶ et le logiciel qui gère les notes dans les IUT !

Les variables Python sont automatiquement créées au moment où on leur assigne une valeur.

Pour choisir le nom d'une variable, il faut respecter certaines règles :

- ▶ le nom doit commencer par une lettre ou par un underscore (non recommandé sauf cas très particuliers)
- ▶ le nom d'une variable ne doit contenir que des caractères alphanumériques courants (pas d'espace dans le nom d'une variable, ni de caractères spéciaux comme les caractères accentués ou tout autre signe comme le signe –)
- ▶ on ne peut pas utiliser certains mots qui possèdent déjà une signification spéciale dans Python (mots réservés).

```
a_1 = 'Bonjour'
b = "Salut"
phrase = "Lycée Charles Carnus"
z = 10
x5 = 3.14
A, B = 5.41, 12
C = D = 3
E = 5 + 16
D = A + E
liste1 = [] #Liste vide
liste2 = [1, 4, 9] #Liste de 3 éléments
liste3 = [[1, 2], [3, 4]] #Liste
t = 1, 5, 'Bonjour' #Tuple
np.arange(3) #array([0, 1, 2])
np.arange(3,6) #array([3, 4, 5])
np.arange(3,7,2) #array([3, 5])
np.array([1, 2, 3]) #array([1, 2, 3])
np.array([[1, 2], [3, 4]]) #array([[1, 2], [3, 4]])
```

Les commentaires

```
# Ceci est un commentaire qui sera ignoré à l'exécution

"""
Ceci est un commentaire
sur plusieurs lignes
"""
```

Programmation de base : Déclaration de variables

```
Z = np.linspace(0,1,11,endpoint=True)  
print(Z)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

```
Z = np.linspace(0,1,11,endpoint=False)  
print(Z)
```

```
[0. 0.0909 0.1818 0.2727 0.3636 0.4545 0.5455 0.6364 0.7273 0.8182 0.9091]
```



```
>>> word = 'Python'
>>> word[0]    # caractère en position 0
'p'
>>> word[5]    # caractère en position 5
'n'
```

Les indices peuvent également être négatifs, on compte alors en partant de la droite. Par exemple :

```
>>> word[-1]
'n'
>>> word[-2]
'o'
>>> word[-6]
'p'
```

```
>>> word[0:2]    # caractères de la position 0 (incluse) à 2 (exclue)
'Py'
>>> word[2:5]    # caractères de la position 2 (incluse) à 5 (exclue)
'tho'
```

Les valeurs par défaut des indices de tranches ont une utilité ; le premier indice vaut zéro par défaut (c.-à-d. lorsqu'il est omis), le deuxième correspond par défaut à la taille de la chaîne de caractères.

```
>>> word[:2]    # caractères du début jusqu'à la position 2 (exclue)
'Py'
>>> word[4:]    # caractères de la position 4 (incluse) jusqu'à la
                fin
'on'
```

On note que le début est toujours inclus et la fin toujours exclue. Les chaînes de caractères, en Python, ne peuvent pas être modifiées. On dit qu'elles sont immuables. Affecter une nouvelle valeur à un indice dans une chaîne produit une erreur.

La fonction native `len()` renvoie la longueur d'une chaîne :

```
>>> s = 'LycéeCharlesCarnus'
>>> len(s)
18
```

```
>>> print('C:\home\name')  
# \n nouvelle ligne
```

C:\home
ame

```
>>> print(r'C:home\name')  
# notez r avant simple quote
```

C:\home\name

```
>>> 'Un' + 'Deux' + 'Trois'
```

'UnDeuxTrois'

```
>>> 3 * 'Un' + 'Deux'
```

'UnUnUnDeux'

```
>>> print("""\n  
Utilisation: machin [OPTIONS]  
    -h            Afficher le message  
    -H hostname   Hôte auquel se connecter  
""")
```

```
Utilisation: machin [OPTIONS]  
    -h            Afficher le message  
    -H hostname   Hôte auquel se connecter
```

Syntaxe

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Paramètres

- objects** objet(s) à l'imprimer. * indique qu'il peut y avoir plus d'un objet
- sep** les objets sont séparés par sep. Valeur par défaut: ' ' (espace simple)
- end** end est imprimé à la fin
- file** doit être un objet avec la méthode `write(string)`. S'il est omis, `sys.stdout` sera utilisé pour imprimer les objets à l'écran.
- flush** si True, le flux est vidé de force. Valeur par défaut : False

Exp : `a = 5; print("a =", a, sep='>', end='\n')`

`a ==>5`

Programmation de base : `print()`

```
import os  
print('Bonjour, ' + os.getlogin() + '! Comment allez-vous ?')
```

Bonjour, utilisateur! Comment allez-vous ?

```
print('home', 'user', 'documents', sep='/')
```

home/user/documents

```
a = (4500 + 2575) / 14800  
print("Le résultat est", a)
```

Le résultat est 0.4780405405405405

```
print(f"Le résultat est {a:.2f}")
```

Le résultat est 0.48

```
print(f"Le résultat est {a:.5f}")
```

Le résultat est 0.47804

Programmation de base : `input()`

`input()` est une fonction intégrée utilisée pour lire une chaîne à partir d'une entrée standard (dans la plupart des cas, le clavier).

Syntaxe

`x = input(prompt)`

où `prompt` est une chaîne (facultative), qui est imprimée sur la sortie standard, avant que la fonction `input()` ne commence à lire la valeur entrée par l'utilisateur. `prompt` est utile pour permettre aux utilisateurs de savoir quel type d'entrée l'application attend.

La fonction `input()` renvoie la chaîne saisie par l'utilisateur dans l'entrée standard.

```
x = input('Saisir une valeur : ')
print('La valeur est : ', x)
type(x)
```

```
Saisir une valeur : 29
La valeur est : 29
str
```

Remarque : La valeur retournée par `input()` est de type "string"

Lire un nombre à l'aide de `input()`

Par défaut, la fonction `input()` renvoie une chaîne de caractères. Si on souhaite lire un nombre, on peut transtyper la chaîne en `int`, `float` ou `complex`, en utilisant respectivement les fonctions `int()`, `float()` et `complex()`.

```
x = int(input('Entrer un entier : '))  
print('La valeur saisie est : ', x)  
y = float(input('Entrer un réel : '))  
print('La valeur saisie est : ', y)
```

```
Entrer un entier : 29  
La valeur saisie est : 29  
Entrer un réel : 3.14  
La valeur saisie est : 3.14
```

```
x = input()  
print('Entrée saisie est : ', x)
```

```
87  
Entrée saisie est : 87
```

Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrite comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
>>> squares[0]
1
>>> squares[-1]
25
>>> squares[-3:]
[9, 16, 25]
>>> squares[:]
[1, 4, 9, 16, 25]
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Les listes sont muables : il est possible de modifier leur contenu.

```
>>> cubes = [1, 8, 27, 65, 125]
>>> cubes[3] = 64
>>> cubes
[1, 8, 27, 64, 125]
```


Opérateurs mathématiques

Symbole	Opération	Types	Exemples
+	Addition	Entier, réel, chaîne de caractères	$3 + 2 \rightarrow 5$ $"a" + "b" \rightarrow "ab"$
-	Soustraction	Entier, réel	$3 - 2 \rightarrow 1$
*	Multiplication	Entier, réel, chaîne de caractères	$3 * 2 \rightarrow 6$ $3 * "b" \rightarrow "bbb"$
**	Puissance	Entier, réel	$3 ** 2 \rightarrow 9$
/	Division	Entier, réel	$3 / 2 \rightarrow 1.5$
//	Division entière (Euclidienne)	Entier, réel	$7 // 2 \rightarrow 3$
%	Modulo (reste de la division)	Entier, réel	$7 \% 2 \rightarrow 1$

Opérateur	Signification
<code>pow(x,y)</code>	x à la puissance y . Par défaut : $0^0 = 1$
<code>divmod(x,y)</code>	Renvoie le couple $(x//y, x\%y)$
<code>abs(x)</code>	Valeur absolue d'un réel, module d'un complexe
<code>int(x)</code> <code>float (x)</code>	Conversion vers le type <code>int</code> , conversion vers le type <code>float</code>
<code>round(x)</code> <code>round(x,n)</code>	Arrondit à l'entier le plus proche, valeur approchée à 10^{-n} près
<code>z.real</code> , <code>z.imag</code> , <code>z.conjugate()</code>	Partie réelle, partie imaginaire, conjugué
<code>complex(x,y)</code> , <code>x+u*1j</code>	Renvoie le nombre complexe $x + iy$

Opérateurs logiques

Opérateur	Opération	Exemples
<i>or</i>	Ou logique	$x \text{ or } y \rightarrow \text{False}$ $x \text{ or } z \rightarrow \text{True}$
<i>and</i>	Et logique	$x \text{ and } y \rightarrow \text{False}$ $x \text{ and } z \rightarrow \text{True}$
<i>not</i>	Non logique	$\text{not } y \rightarrow \text{False}$ $\text{not } z \rightarrow \text{True}$

Opérateurs de comparaison

Opérateur	Opération
<	Strictement inférieur (<)
<=	Inférieur ou égal (\leq)
>	Strictement supérieur (>)
>=	Supérieur ou égal (\geq)
==	égal
!=	Différent (\neq)

Programmation de base : Les opérateurs

```
2+11*5
```

57

```
# Ecrire un commentaire
```

```
print('Bonjour')
```

Bonjour

```
print(5.6/2)
```

2.8

```
# Calcul simple.  
print("2+3*5 donne :")  
print(2+3*5)
```

2+3*5 donne :
17

```
# Affectation  
a=5  
print(a)
```

5

```
a == 4
```

False

```
x, y = (1, 2) # ou  
x, y = [1, 2] # ou  
(x, y) = (1, 2)  
print(x)  
print(y)
```

1
2

```
abc = "Deux mots"  
print(abc)
```

Deux mots

```
a = 7  
print("double de ", a, " est ", 2*  
      a)
```

double de 7 est 14

Les structures de contrôles

Test if

```
#Test de parite  
  
nb=3  
if (nb%2)==0:  
    print("Nombre pair")  
else:  
    print("Nombre impair")
```

Pas de switch

Boucle for

```
#boucle for allant de 0 a 9  
  
for indice in range(10):  
    #affichage  
    print("indice=%d" % indice)
```

Boucle while

```
#Boucle while allant de 0 a 9  
  
indice=0  
while(indice <10):  
    print("indice=%d" % indice)  
    indice=indice+1
```

Condition *if*, *elif*, *else*

Exemple 1 :

```
if a == 4:  
    print('Fin')
```

Exemple 2 :

```
a = 5  
  
if a == 1:  
    print(1)  
elif a == 2:  
    print(2)  
else:  
    print('Fin')
```

Boucle *for*

Exemple 1 :

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

Exemple 2 :

```
for word in ('juin', 'juillet', 'août'):  
    print("Un mois d'été : %s" % word)
```


Exemple 1 :

```
x = 5
def plusCinq(y):
    return x + y

plusCinq(9)
```

Exemple 2 :

```
def triple(x):
    return 3 * x

triple(15)
```

Exemple 3 :

```
def triple1(x=2):
    return 3 * x

a=triple1()
print(a)

b=triple1(5)
print(b)
```

Exemple 4 :

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def f(x):
    return x**2 + 10*np.sin(x)

x = np.arange(-8, 8, 0.1)
plt.plot(x, f(x))
```

Programmation de base : Les courbes

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

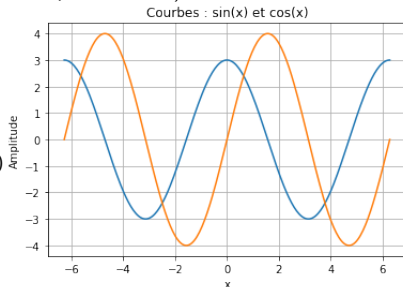
```
X = np.linspace(-2*np.pi, 2*np.pi, 256, endpoint=True)
```

```
C, S = 3*np.cos(X), 4*np.sin(X)
```

```
plt.figure()
plt.plot(X, C)
plt.plot(X, S)
```

```
plt.title('Courbes : sin(x) et cos(x)')
plt.xlabel('x')
plt.ylabel('Amplitude')
```

```
plt.grid()
plt.show()
```



Un nombre complexe z peut s'écrire sous plusieurs formes.

$$z = a + jb = M \cos(\theta) + jM \sin(\theta) = M(\cos(\theta) + j \sin(\theta)) = Me^{j\theta}$$

Avec a la partie réelle (notée $\Re e$), b la partie imaginaire (notée $\Im m$), j le nombre imaginaire pur (noté i en mathématiques).

M le module de z et θ l'argument de z .

Le conjugué du nombre complexe z est noté \bar{z} , avec $\bar{z} = a - jb = Me^{-j\theta}$

Déclaration d'un nombre complexe sous Python

```
import numpy as np
import matplotlib.pyplot as plt
z = 5+3j
r = np.real(z)
i = np.imag(z)
print("Partie réelle =",r," , Partie imaginaire =",i)
m = np.abs(z)
p = np.angle(z)
print("Module de z =",m," , Argument de z =",p)
```

Partie réelle = 5.0 , Partie imaginaire = 3.0

Module de z = 5.830951894845301 , Argument de z = 0.5404195002705842