ESISA

Notes de cours Python

M. Lahmer

1. Syntaxe de base 3
    1. Variables, assignation 3
    2. Input / Output 3
    3. Les opérateurs 4
    4. Les blocs et les commentaires 4
2. Structures de contrôle 4
    1. les boucle for et while 5
    2. break, continue 5
3. Les chaines de caractères 5
    1. Les fonctions de base 5
    2. L'indexation et parcours 6
4. Listes, tuples, ensembles et dictionnaires 7
    Les listes 7
    Les tuples 8
    Les ensembles 8
    Les dictionnaires 8
5. Les Fonctions 9
    Préliminaires 9
    Portée d'une variable 9
6. Les fichiers 10
7. Scripting avec Python 10
    Gestion des fichiers 10
    Execution des commandes shell 11
    Les processus sous python 11
    Instructions de Mise en attente 11
    Les pipes 12
    La redirection 13
8. Ploting avec python 13

1. Syntaxe de base

C'est un langage interprété, objet mais il peut être utilisé en tant que langage de script

Un commentaire Python commence toujours par le caractère spécial **#**

#/usr/bin/python

Les différents types des variables sont donnés par :

- Les données numériques sont de type **int**, **long** ou **float** suivant la valeur que vous utiliser
- Les chaînes de caractères délimitées par " " ou ' ' ou """ """ sont de type **str**
- Les booléens qui vaudront **True** ou **False**

1. Variables, assignation

Python est un langage dynamiquement typé, les variables n'ont pas besoin d'être déclarées, et leur type peut changer au cours de l'exécution. la fonction type() permet de savoir le type d'une variable

```
>>>a = 3
>>>type(a)
<type 'int'>
>>>a = '3'
>>>type(a)
<type 'str'>
```

```
>>>x=3.4
>>>type(x)
<type 'float'>
>>>b=6L
>>>type(b)
<type 'long'>
```

```
>>>3443256367048874627473999435732 75**20 #** est l operateur puissance
54876598571325518594229020652037407810134044971784202365102816906460768010
75544744453717760365148594670214858912192873964753226919864646159434857463
19235702156543512417751028955191028353050730398385128281188168428548313235
06494171195566550312861942071834318984774912763694202700348365990057644564
24291668950662109074313149119821754815742349365677428653280901673871458534
65614042614379568436182532558460975757139559987167497637040786080670976674
91147401388773390897343862558644505267809146444731936008700325689058599508
41475585957459044641792979347389309661545460773830117354737529378479425950
40121432078809696341558088712199605652131140232086181640625L
```

La taille des entiers n'est limitée que par la mémoire de la machine

2. Input / Output

Affichage

```
print ("Demo x!=", 6)
```

on peut utiliser un affichage formaté en utilisant les formats classiques (int:%d),(float:%f),(bool et str:%s)

```
print ("My name is %s and weight is %d kg!" %'Mohamed', 76)
```

ou en utilisant la fonction format :'{0} {1}..{n}'.format(v0,v1,..vn)

```
print ("My name is {0} and weight is {1} kg!".format('Mohamed', 76))
```

Lecture

```
>>>X=input( "Donner une valeur : " )
```

Il faut saisir un type prédéfini python (int, float, string, bool,…)

3. Les opérateurs

les opérateurs arithmétiques sont (+, -, *, /, // (quotient), % (reste de division) et ** (puissance)).
L'ordre de calcul d'une expression arithmétique tient compte de la priorité de chaque opérateur

(+,-) (*, /, //) (**)

```
>>> 5 – 8 + 4 * 2 ** 3
29
```

(5-8) + (4 * (2 ** 3))

les opérateurs logiques sont (==,!=, <,<=, >, >=, and, or, not). L'opérateur **in** permet de tester si une valeur appartient à une collection (chaîne de caractères, liste).

```
>>> s = 'SALAM'
>>> 'A' in s
True
```

Les opérateurs binaires permettent d'effectuer des opérations bas niveau en binaire sur des entiers

>>	décalage à droite	7 >> 1 vaux 3	111 >> 2 = 011
<<	décalage à gauche	8 << 2 vaux 32	1000 << 2 = 100000
&	le et binaire	5 & 3 vaux 1	101 & 011 = 001
^	le xor	5 ^ 3 vaux 6	101 ^ 011 = 110
|	le ou binaire	5 | 3 vaux 7	101 | 011 = 111
~	le complément à 1	~ 3 vaux -4 (-3 -1)	~ 011= - 100

4. Les blocs et les commentaires

L'indentation en python a une valeur syntaxique : elle sert à délimiter les blocs. Toutes les lignes d'un même bloc doivent être précédées du même nombre d'espaces blancs. Les retours à la ligne sont interdits en Python, sauf entre parenthèses (), crochets [], et accolades {}.

```
if (a == b
    and c == d):  print yes'
```

Un commentaire sur une seule ligne est précédé par **#**
Un commentaire sur plusieurs lignes ''' ligne commentaire 1
ligne commentaire 2 '''

2. Structures de contrôle

La seule construction conditionnelle existante en Python est **if... elif... else....** Toutes les branches sont optionnelles, à l'exception du if, il peut y avoir un nombre quelconque de elif, mais un seul else à la fin.

```
if a == b == c:
    print 'égaux'
elif a <= b <= c  or  c <= b <= a:
    print 'b au milieu'
elif b <= a <= c  or  c <= a <= b:
    print 'a au milieu'
else:
    print 'c au milieu'
```

sous python il n'existe pas de switch case mais on peut faire mieux avec le données de type dictionnaire

1. les boucle for et while

Il existe deux types de boucles en Python. La plus couramment utilisée est le for... in qui permet de parcourir les éléments d'une liste.

```
for i in range(10):
    print i
```

La boucle for est souvent utilisée en conjonction avec la fonction **range**, dont la syntaxe générale est : **range(start, end, step) a**insi appelée, la fonction génère la liste des entiers entre start (inclus) et end (non inclus) avec pas de step :

```
>>range(0, 10, 2)
[0, 2, 4, 6, 8]
```

Les deux autres syntaxes admissibles sont range(start, end) (pas égal à 1) et range(end) (début égal à 0).

La deuxième boucle est très similaire à la boucle while en C.

```
a = 0
while a < 10 :
    a = a + 1
    print a
```

2. break, continue

Comme en C, l'instruction break sort de la boucle sans vérifier la condition :

```
for i in range(10):
    if i > 2 :
        break
    print i
0 1 2
```

```
for i in range(10):
    if i % 2 == 0:
        continue
    print i
1 3 5 7 9
```

L'instruction continue passe l'itération suivante en sautant le reste du corps. Les chaînes de caractères

3. Les chaines de caractères

1. Les fonctions de base

la taille d'une chaîne de caractère n'est limitée que par la taille de la mémoire physique et l'architecture (32 ou 64 bits) de votre machine. Les chaînes de caractères délimitées par " " ou ' ' ou """ """ sont de type **str**

```
>>>str1 = "this is string example...."
>>>str2 = "exam";
La longueur est donnée par len
>>>len(str2)
4
La concaténation se fait par +
>>>str1+str2
this is string example...exam
La répétition
>>>str2*3
examexamexam
```

fonction	rôle	exemple
strip()	élimine les espaces	' salam '.strip()—>'salam'
find (motif)	cherche motif dans une chaine et revoie sa position sinon -1	'dev python'.find('py') —> 4
startswith(prefix)	True si la chaine commence par prefix	'salam'.startswith('z')—> False
endswith(suffixe)	True si la chaine se termine par suffixe	'*.py'.endswith('py') —> True
replace(substr,anotherstr)	remplace substrat par anathoerstr dans une chaine	'*.py'.replace('py','bakup')
isdigit() isalpha() isalnum() isspace()	permet de tester si une chaine est numérique, alphabétique, alphanumérique ou c'est des espaces	'1234'.isdigit() —> True 'A111'.isalnum() —> True 'AAAA'.isalpha() —> True 'A1 AA'.isspace() —> False
upper() lower()	Permet de transformer en Majuscule respect. en Minuscule	'Salam'.upper() —> 'SALAM' 'Salam'.lower() —> 'salam'
count(sub)	retourne la fréquence de la sous chaine sub dans une chaine	'salam'.count('a')—>2
str (objet)	converti l'objet en une chaine	str(13.0) —> '13.0'
split(separateur)	Permet de décomposer la chaine dans un tableaux selon le séparateur (par défaut l'espace)	'ceci est une demo'.split() —> ['ceci','est','une','demo'] 'login:pass'.split(':') —> ['login','pass']

A noter que une chaîne de caractère sous python n'accepte pas l'affection indexée : s='ESISA', s[0] = 'A' n'est pas autorisé

2. L'indexation et parcours

L'indexation

Pour indexer une chaîne on utilise l'opérateur [] dans lequel l'index est un entier qui peut être négatif.
En tant que programmeur, je vous déconseille d'utiliser les indexes négatifs c'est du n'importe quoi. Ainsi au lieu de travailler avec -i vous pouvez travailler avec len(s)-i.

s='prophete Mohamed'

p	r	o	p	h	e	t	e		M	o	h	a	m	e	d
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Dans cet exemple len(s)=16. s[0] vaux 'p', s[8] vaux ' ' et s[-6]=s[len(s)-6]=s[10] vaux 'o'.
Attention une chaîne est un objet non mutable :on ne peut pas faire l'affectation par les indexes s[0]='Z' est interdite
Pour l'extraction d'une sous chaîne on utilise trois indexes [d:f:p] ou d,f est p sont respectivement début, fin et pas d'extraction (par défaut d=0,f=16 et p=1)

```
>>>s[9:16:1]
'Mohamed'
```

```
>>>s[::]
'prophete Mohamed'
```

```
>>>s[::3]
'pptMad'
```

Pour inverser un chaîne, en commence par la fin (ici 15) en utilisant un pas égale à -1.

```
>>>s[15::-1]
'demahoM etehporp'
```

Le parcours
Il existe deux manières pour parcourir une chaine

```
for c in s: print c
```

```
for index in range(len(s)): print s[index]
```

4. Listes, tuples, ensembles et dictionnaires

La longueur, l'indexation et le parcours sont identique à celui des chaînes de caractères
Les listes
L'un des objets les plus utilisés en Python, ce sont les listes. On déclare une liste avec les crochets [], et on accède à ses éléments de la même façon qu'on accède aux éléments d'une chaine

```
>>>L = [1, 2, 'a', True]
>>>L
[1, 2, 'a', True]
```

l'extraction se fait de la même manière que dans les chaines de caractères (voir 3.2)

```
>>> L[2::]
['a',True]
```

Pour initialiser une liste L=[] et l'ajout se fait de la même façon que les string en utilisant l'opérateur +. A titre d'exemple pour construire une liste contenant les nombres paires de 1 à 100

```
L = [ i for i in range(1,101) if i % 2 == 0 ]
```

```
L=[]
for i in range(1,101):
      if i % 2 == 0:
            L+=[i]
```

```
liste des diviseur de n
[d for d in range(1,n+1) if n % d ==0]
```

Quelques méthodes prédéfinies

append(elt)	Ajoute un élément est à la liste	L=[] L.append(123)
index(elt)	renvoie l'indexe de la première occurence de let	L=[12,3,44,3,5,77] L.index(3) —->1
L.remove(elt)	supprime de elt de la liste	L=[12,3,44,3,5,77] L.remove(3) —->L devient [12,44,3,5,77]
L.reverse()	inverse les places des éléments d'une liste	L=[True,[1,2]] L.reverse() —->L devient [[1,2],True]
L.sort()	permet de trier le contenu d'une liste. Attention les éléments de la liste doivent être naturellement comparable	L=["z", "d", "a"] L.sort() —> L devient ["a", "d", "z"]
len(L),min(L), max(L), sum(L)	longueur, min, max ou la somme des éléments d'une liste (même condition que ce qui précède)	L=[23,1,4,2] min(L)—>1 max(L) —->23 sum(L) —>30

Les tuples

Un tuple est une liste non modifiable c'est les types énumérés en java. On les utilise pour définir les jours de la semaines, le mode de payement, les couleurs,… On ne peut pas faire l'affectation directe

couleur=("red", "green", "blue")
couleur[1]—> "green"
len (couleur) —> 3
pour supprimer un tuple on utilise le mot clé del
del couleur

Les ensembles

Un set est une liste d'elements uniques non ordonnés. On peut construire un set à partira d'une liste.

S={1,2,3} ou T=set([2,2,3,6,7])
S & T donne l'intersection de S et T ici c'est l'ensemble {2,3}
S | T donne l'union de S et T {1,2,3,6,7}
S - T les éléments de S qui ne sont pas dans T {1}
S <= T retourne True si S est inclus dans T False sinon

Les dictionnaires

Un dictionnaire est une liste d'objets ou chaque objet est composé d'une clé et d'une valeur. La clé et la valeur sont concatenés par :

tele = {'directeur':112,'secretaire':113,'personnel':200}
tele['secretaire']—>113
tele.keys() —->['directeur','secretaire','personnel']
tele.values() —> [112,113,200]
Si on veut parcourir à la fois les clés et les valeurs :

```
for k,v in tele.items() :
        print (k,v)
```

pour ajouter un élément au dictionnaire tele['it']=115, pour ajouter plusieurs éléments
tels.update({'finance':120,'dev':117})

5. Les Fonctions

Préliminaires

Les fonctions Python sont définies par le mot clef **def**. Elles peuvent prendre un nombre arbitraire de paramètres, et renvoient une valeur à l'aide du mot clef return. Toute fonction renvoie une valeur, les fonctions qui n'ont pas de return renvoient la valeur spéciale None.

```
def max(x, y):
    if x > y:
        return x
    else:
        return y
```

Certains paramètres peuvent prendre des valeurs par défaut. Si un paramètre prend une valeur par défaut, tous ceux qui le suivent doivent aussi en prendre.

```
def pow(x, n=0):
      return x**n
>>>pow(3) # quand on ne donne pas n, par defaut c est 0
1
>>>pow(3,2)
9
```

Une fonction sous python peut prendre un nombre d'arguments variable *args

```
def prod(*args):
    pr=1
    for p in args:
        pr*=p
    return pr
print (prod(2,3))   #Affichera 6
print (prod(2,3,4)) #Affichera 24
Il existe aussi les fonctions à paramètre clés (l'ordre n'est pas important) :
def process(**args):
     for k,v in args.items():
          print(k,v)
process('sys='win','version'='7')
```

Portée d'une variable

une variable initialisée à l'intérieur d'une fonction est une variable locale à la fonction.

```
var=10
def demo() :
      var=3
print (var) # on aura toujours 10
```

Pour dire à python que var de demo est une variable globale il faut ajouter le mot clé global.

```
def demo() :
      global var
      var=3
print (var)# on aura 3
```

6. Les fichiers

La fonction open(fichier,mode) permet d'ouvrir un fichier selon un mode (r, w, a, r+). Sous windows, il y a une distinction entre les fichiers binaires et texte ainsi pour les fichiers binaires en retrouve en plus les modes (rb, wb et r+b).

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
```

La fonction open retourne un objet qui dispose des méthodes pour l'écriture, la lecture, le positionnement et la fermeture.

On crée un fichier vide dans le répertoire courant

```
>>> file=open("demo.txt","w")
```

On écrit une deux lignes

```
>>> file.write("Ce ci est un test \n123456789")
```

On ferme le fichier puis on l'ouvre en mode lecture écriture

```
>>> file.close();
>>> file=open("demo.txt","r+")
```

Pour lire la totalité du fichier

```
>>> print (file.read())
                    Ce ci est un test
                    123456789
```

Pour revenir au début, on utilise la fonction seek(offset)

```
>>> file.seek(0)
```

On peut lire un nombre de caractères bien définit file.read(nb_char)

```
>>>file.read(5)
                    Ce ci
```

Pour lire une ligne du fichier, on utilise file.readline() sinon pour lire toutes les lignes on utilise la fonction file.readlines() (retourne une liste)

```
>>>for ligne in file.readlines():
        print (ligne)
```

la fonction enumerate permet de renvoyer le numéro de ligne courante dans un fichier texte.

```
>>> for nl,ligne in enumerate(file):
    ... print (nl,ligne)
                    0 Ce ci est un test
                    1 123456789
```

7. Scripting avec Python

Gestion des fichiers

os.mkdir("test") : permet de créer un nouveau répertoire test

os.chdir("test") : c'est cd, elle permet d'accéder au répertoire test

os.getcwd() : retourne le nom du répertoire courant

os.rmdir("/home/user/test") : supprime un répertoire (il doit être vide)

dirs = os.listdir(path) : retourne le contenu d'un répertoire sous forme de liste

os.remove("demo.txt") : supprime un fichier

os.rename(AncienNom,NouveauNom) : renomme un fichier

Les tests de base sur les dossiers et fichiers se font tous à partir de fonctions du module os.path.

os.path.exists(*path*) : Permet de vérifier si path est un chemin réel.

os.path.isabs(*path*) :Teste si path est un chemin absolu.

os.path.isdir(*path*) :Teste si path pointe vers un dossier.

os.path.isfile(*path*) :Teste si path pointe vers un fichier.

os.path.abspath(*path*) : Retourne le chemin absolu du chemin relatif path donné en argument.

os.path.basename(*path*) : Retourne le dernier élément du chemin.

os.path.dirname(*path*) : Retourne le chemin pour accéder au dernier élément.

os.path.getatime(*path*), os.path.getmtime(*path*)¶, os.path.getctime(*path*) : retourne respectivement le dernier temps d'accès, de modification et de changement d'état

os.path.getsize(*path*) : Retourne la taille d'un fichier

Exercices :

script find

Ecrire un script en python permettant d'afficher de façon récursive les fichiers d'un répertoire dont le nom contient un motif.

Execution des commandes shell

Pour exécuter une commande sans traiter le résultat de retour on utilise la fonction system

```
os.system("clear")
```

Sinon on fait appelle à os.popen (commande) qui permet d'exécuter une commande dans un pipe que vous pouvez utiliser à travers une variable.

```
ll=os.popen('ls -l')
for ligne in ll:
 print ligne
ll.close()
```

Les processus sous python

Un processus est un programme en cours d'execution. Un processus passe par plusieurs états avant qu'il se termine :
endormie (S), en exécution (R), suspendu (T) voir zombie (Z)
Pour avoir les caractéristiques de base d'un processus (comme ps linux)
os.getpid() : retourne le pid du processus en cours
os.getppid() : retourne le ppid
os.getuid() : retourne le uid de l'utilisateur propriétaire du processus
os.getgid() : retourne le gid du groupe propriétaire du processus
Pour créer un processus fils on utilise la fonction os.fork(). La fonction fork retourne trois valeurs (0 : le fils, >0 : le père, -1 erreur).

```
import os, sys
pid_fils=os.fork()
if pid_fils == -1 :
      print('erreur creation')
      sys.exit(0)
elif pid_fils == 0 :
      print('Je suis le Fils PID : %d, PPID : %d' %(os.getpid(),os.getppid()))
else :
      print('Je suis le Pere PID : %d, PPID : %d' %(os.gettpid(),os.getppid()))
```

Instructions de Mise en attente

On peut mettre un processus dans l'état endormis en utilisant :

- Une instruction E/S comme input
- Attente temporelle time.sleep(ns) (time.sleep(0.1) attente 100 ms)

- Attente sur la terminaison d'un fils os.wait() (pid,status=os.wait() retourne le pid du fils qui a terminé et l'état de retour associée)
- Attente d'un signal signal.pause()

Pour forcer un père à attendre la fin de son fils en utilisant un signale ici SIGUSR1

```
import os, sys, signal
pid_fils=os.fork()
if pid_fils == -1 :
      print('erreur creation')
      sys.exit(0)
elif pid_fils == 0 :
      print('Je suis le Fils PID : %d, PPID : %d' %(os.getpid(),os.getppid()))
      kill(os.getppid(),signal.SIGUSR1)
else :
      signal.pause()
      print('Je suis le Pere PID : %d, PPID : %d' %(os.gettpid(),os.getppid()))
```

pour redéfinir le traitement du père une fois il reçoit le signale, on ajoute la fonction signal.signal(signal,handler)

```
def handler(num_sig, pile):
      print('Signal recu %d' %(num_sig))
def code_fils(pid):
    print('Traitement du fils : %d' %(pid))
def code_pere():
    print('Traitement du pere')
pid_fils=os.fork()
if pid_fils == -1 :
    print ('erreur creation processus')
    sys.exit(0)
if pid_fils == 0 :
     code_fils(os.getpid())
     os.kill(os.getppid(),signal.SIGUSR1)
else :
      signal.signal(signal.SIGUSR1,handler)
      signal.pause()
```

Les pipes

un pipe (tube) est une structure logique en mémoire avec deux extrémités une pour la lecture et autre pour l'écriture. C'est un moyen de communication unidirectionnel entre processus. Le système conserve temporairement l'information transmise (écrite) jusqu'à ce qu'elle soit lue par le processus de réception.

Pour la communication bidirectionnelle entre les processus, deux pipes peuvent être mis en place, un pour chaque direction. Les processus utilisant des canaux doivent avoir un processus parent commun.

```
import os,sys,signal
r,w = os.pipe()
pid_child = os.fork()
if pid_child == 0 :
```

```
    #Tache du fils
    fp=os.popen('ls -l','r')
    result = fp.read()
    fp.close()
    #Ecriture du resultat
    os.close(r)
    fw = os.fdopen(w,'w')
    fw.write(result)
    fw.close()
    #Fin du fils
    sys.exit(10)
    elif pid_child >0 :
    #Attente la terminaison du fils
    pid,status=os.wait()
    os.close(w)
    #Lire le resultat depuis le pipe
    fr=os.fdopen(r,'r')
    result=fr.read()
    print('Message recu :',result)
    fr.close()
```

La redirection

Chaque processus reconnaît trois descripteurs de fichiers standards à savoir :

- stdin (0) : l'entrée clavier
- stdout (1) : l'écran
- stderr (2) : sortie d'erreur par défaut l'écran

Pour dire à un processus d'écrire dans un fichier au lieu de l'écran ou bien de lire depuis un fichier au lieu du clavier on fait appel aux fonctions systèmes dup et dup2 :

- os.dup2(newfd,oldfd) : permet de remplacer l'ancien descripteur oldfd par newfd

Exemple :

On veut mettre en place un script qui réalise ls -l | grep '.py'. On aura besoin de deux processus (père et fils) et un pipe. Le processus fils exécutera la commande ls -l et au lieu d'écrire le résultat à l'écran il les redirige vers le pipe. Le père (grep '.py') va lire son entrée depuis le pipe au lieu du clavier. Pour exécuter une commande, on va faire appel à os.execvp().

```
import os,sys,signal
r,w = os.pipe()
pid_child = os.fork()
if pid_child == 0 :
    os.close(r)
    os.dup2(w,1)
    os.execvp('ls',['ls','-l'])
elif pid_child >0 :
    #Attente la terminaison du fils
    pid,status=os.wait()
    os.close(w)
    os.dup2(r,0)
    os.execvp('grep',['grep','.py'])
```

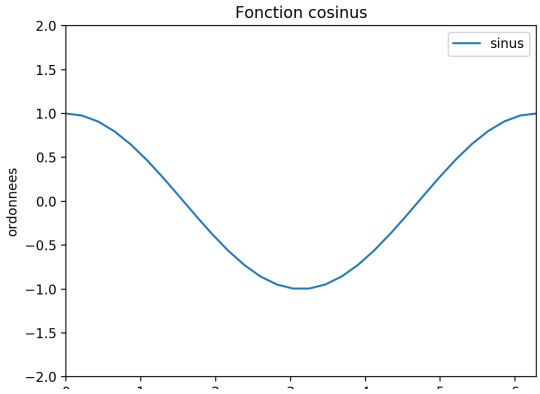
8. Ploting avec python

Il faut installer les bibliothèques matplotlib et numpy (les calcules) :

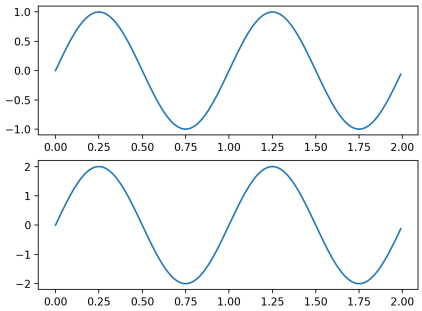
```
sudo apt install python3-pip
```

```
python -m pip install -U pip
python -m pip install -U matplotlib
python -m pip install -U numpy
```

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 30)
y = np.cos(x)
#definition des limites des axes
plt.xlim(0, 2*np.pi)
plt.ylim(-2, 2)
#Titre de la figure
plt.title("Fonction cosinus")
plt.plot(x, y,label="cosinus")
#afficher la légende
plt.legend()
#nom des axes
plt.xlabel("abscisses")
plt.ylabel("ordonnees")
plt.show() # affiche la figure a l'ecran
```




```
On peut dessiner plusieurs courbes dans la même figure pour cela on
utilise subplot
import matplotlib.pyplot as plt
import numpy as np
t = np.arange(0.0, 2.0, 0.01)
s1 = np.sin(2*np.pi*t)
s2 = np.sin(4*np.pi*t)
plt.figure(1)
'''2 le nombre de lignes, 1 le numéro
de la colonne et 1 l'indexe de la figure
'''
plt.subplot(211)
plt.plot(t, s1)
plt.subplot(212)
plt.plot(t, 2*s1)
plt.show()
```




```
Pour dessiner un histogramme
from matplotlib import pyplot as plot
import numpy as np

# variable gaussienne
data = np.random.randn(1000)
# plot normed histogram
plot.hist(data)
plot.show()
```