Introduction à l'informatique

Cours 10

kn@lmf.cnrs.fr
https://usr.lmf.cnrs.fr/~kn

université
PARIS-SACLAY

Plan

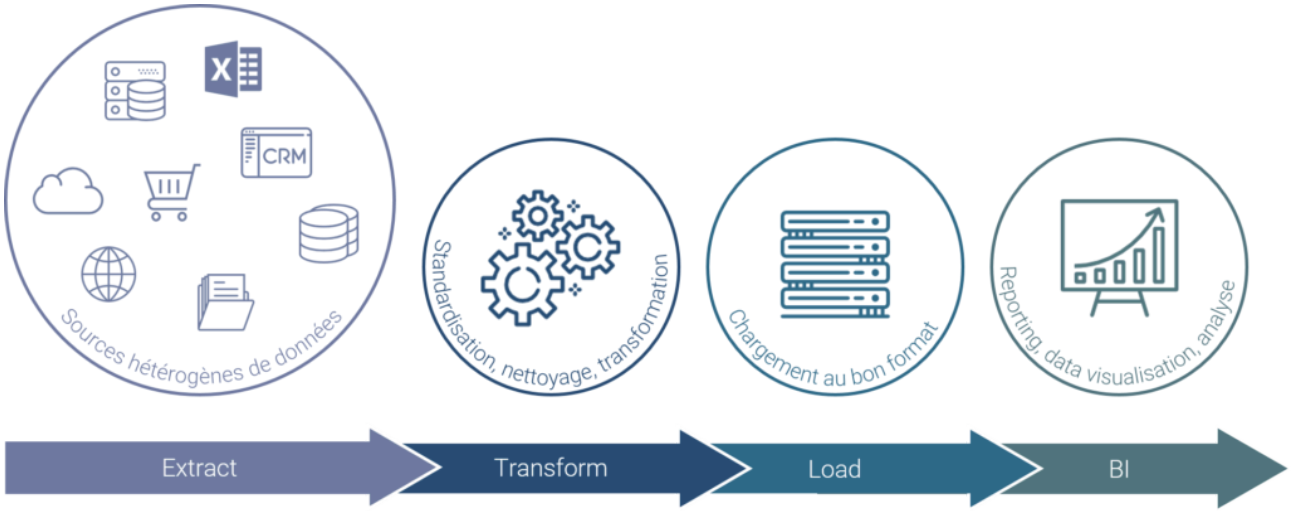
1 Présentation du cours ✓
2 Le système Unix ✓
3 Le système Unix (2) ✓
4 Python (1) : expressions, types de bases, if/else ✓
5 Python (2) : boucles, tableaux, exceptions ✓
6 Python (3) : Textes, chaînes de caractères, entrées/sorties ✓
7 Python (4) : Fonctions ✓
8 Python (5) : Concepts avancés ✓
9 Applications (1) : Introduction aux réseaux ✓
10 Applications (2) : HTTP/HTTPS/HTML/CSS ✓
11 Traîtement de données
- 11.1 Lecture de fichiers de données
- 11.2 Compréhensions de listes
- 11.3 Graphiques avec matplotlib

Traîtement de données ?

La manipulation de données se décompose en trois grandes phases

◆ *Extraction* : les données sont extraites des sources brutes (fichiers textes, bases de données, capteurs, fichiers excel, ...)
◆ *Transformation* : les données extraites sont préparées (validées, corrigées, regroupées logiquement, ...)
◆ *Chargement* (Load): les données, dans leur format final sont enregistrées à destination (autre fichier, base de données).

Un processus métier peut ensuite consommer les données.



©axysweb.com

Fichier textes

Format universel d'échange de données

- Fichier texte simple
- Fichier CSV (« colonnes » séparées par des virgules)
- Fichier XML (« éléments » structurés par des **`<balises></balises>`**)
- Fichier JSON (« éléments » structurés par des **`{ "nom" : "valeur" }`**)

Comment est représenté l'information dans un fichier texte ?

C'est une simple suite d'octets. Les octets sont groupés par paquets (pas forcément un par un) pour former un point de code (un entier)

Cet entier correspond au code du caractères (cf. le cours sur **`UTF-8`** et **`unicode`**)

Retour sur le type `str`

Les chaînes de caractères en Python sont les objets de la classe **`str`**. Les chaînes sont non modifiables (on ne peut pas modifier le $i^{ème}$ caractère).

`str(v)` : transforme n'importe quelle valeur **`v`** en une chaîne
`len(s)` : renvoie la longueur d'une chaîne (nombre de caractères)
`s``.split(c)` : sépare la chaîne **`s`** selon le séparateur **`c`**
`s``.splitlines()` : sépare la chaîne en tableau de lignes (selon les caractères de fin de ligne comme **`\n`**)
`s``.upper()/.lower()` : convertit la chaîne en majuscule/minuscule
`s``.capitalize()` : transforme le premier caractère en majuscule
`s``.strip()` : retire les blancs en début et fin de chaîne
`s``.replace(motif, c, n)` : remplace les **`n`** premières occurrences de la sous-chaîne **`motif`** par la chaîne **`c`**. Si **`n`** est absent, remplace toutes les occurrences.

Les chaînes de format

L'opérateur **%** est défini pour les chaînes de caractères. Il permet d'insérer des valeurs dans une *chaîne de format*

```
age = 42
nom = "toto"
message = f"L'an prochain, {nom.capitalize()} aura {age + 1} ans"

print(message)
#Affiche L'an prochain, Toto aura 43 ans
```

On peut utiliser des chaînes **f'...'**, **f""** ou **f""" ... """**

Lire et écrire des fichiers

L'interface reflète la manière dont les systèmes d'exploitation représente les fichiers :

- Les fichiers sont ouverts selon un mode (lecture, écriture, ...)
- Un fichier ouvert est représenté par un *descripteur de fichier*
- Un descripteur représente une position courante (ou un curseur) dans le fichier
- On peut lire/écrire des octets à la position courante et se déplacer
- On peut (on doit) fermer un descripteur de fichier après l'avoir utilisé

Noms de fichiers

Les fichiers sont dénotés par leur nom et leur chemin, c'est à dire la liste des dossiers à traverser pour trouver le fichier.

◆ Sur les systèmes Unix (Linux, MacOS, Android, iOS, ...), un dossier unique, nommé **`/`** contient tous les autres. Les chemins sont séparés par des **`/`** :

```
/home/kim/Documents/cours/notes.txt
```

◆ Sous Windows, il y a un répertoire racine par périphérique de stockage (**`C:`**, **`D:`**, **`Z:`** **`\\domain`**). Les chemins sont séparés par des **`\`** :

```
C:\Users\kim\Mes Documents\cours\notes.txt
```

Pour du code écrit dans l'environnement Jupyter, on utilise la convention *Unix*

Ouverture d'un fichier

```
f = open (chemin, mode)
```

chemin est une chaîne de caractères. **mode** est l'une des chaînes :

"r" : ouverture en lecture seule
"w" : ouverture en écriture seule, écrase le fichier existant
"w+" : ouverture en lecture et écriture, écrase le fichier existant
"a" : ouverture en écriture seule, ajoute en fin du fichier existant
"a+" : ouverture en lecture et écriture, ajoute en fin du fichier existant

Les données renvoyées par les lectures et utilisées pour les écritures doivent être de type **str**

On peut rajouter un **b** comme deuxième lettre du mode (**"rb"**, **"wb+"**, ...). Les fichiers sont alors ouvert en mode binaire et les données lues et écrites sont de type **bytes**.

Lectures et écritures dans un fichier

Une fois un fichier ouvert on dispose des méthodes suivantes :

`f.read(n)` : Lit au plus **`n`** caractères ou octets (en fonction du mode) à partir de la position courante, renvoyés comme un **`str`** ou un **`bytes`**. Si **`n`** est absent, renvoie tout le fichier.

`f.readline()` : Lit une ligne de texte à partir de la position courante

`f.readlines()` : Lit toutes lignes à partir de la position courante, renvoyées dans un tableau

`f.write(s)` : Écrite la chaîne **`s`** à partir de la position courante.

`f.tell()` : Renvoie la position courante (nombre de caractères ou d'octets) à partir du début du fichier.

`f.seek(n, o)` : Déplace le curseur de **`n`** caractères ou octets. Si **`o`** vaut 0, déplace à partir du début du fichier, si **`o`** vaut 1, déplace à partir de la position courante et si **`n`** vaut 2, à partir de la fin du fichier.

Fermeture du fichier

Il faut toujours fermer les fichiers ouverts.

◆ Les systèmes d'exploitation imposent une limite au nombres de fichiers ouverts en même temps par un utilisateur
◆ Pour les fichiers ouvert en écriture, si le fichier n'est pas fermé certaines écritures peuvent être perdues.

`f.close()` : ferme un fichier ouvert. Provoque une erreur si **`f`** est déja fermé.

Exemple complet

Ouvrir un fichier **`fichier.txt`** et créer une copie où chaque ligne est précédée de **`Ln:`** où *n* est le numéro de la ligne

```
f = open("fichier.txt", "r")
lignes = f.readlines()
f.close()

f = open("fichier_num.txt", "w")
for i in range(0, len(lignes)):
    f.write(f"L{i}: {lignes[i]}")
f.close()
```

Arguments nommés

En Python, il est possible de définir des fonctions ayant des *arguments nommés* ayant une valeur par défaut. Si ces fonctions sont appelées sans donner ces arguments, leur valeur par défaut est utilisée. Sinon il faut utiliser la syntaxe **`nom=valeur`**

Exemple : la fonction **`print(…)`** attend un nombre arbitraire d'arguments et un argument nommé **`end`** affiché en fin de ligne et valant **`'\n'`** par défaut.

```
print("A")
print("B")
# Affiche
# A
# B

print("A", end='') # on n'affiche pas de retour à la ligne.
print("B")
# Affiche
# AB
```

Format CSV

On appelle format CSV tout format de fichier texte encodant des données tabulées. Avec les contraintes suivantes:

- Une ligne du fichier représente une ligne de la table
- Les champs d'une ligne sont séparés par des virgules

Cette définition simple est cependant approximative

- Certains fichiers utilisent d'autres séparateurs («; », « : » , ...)
- Certains fichiers comportent une ligne d'en-têtes de colonnes
- Certains fichiers mettent des guillemets droits (") autour des champs de type texte
- Certains champs de texte peuvent contenir, des virgules ou des retour chariot

Les fichiers CSV sont souvent difficile à lire quand ils ont été mal construits

Chargement de CSV en Python

Les fonctionalités liées au fichiers CSV sont dans le module **csv**

```
import csv
#on veut lire un fichier "fichier.csv" contenant
# Nom;Note
# Toto;14
# Titi;13
# Tata;18
f = open("fichier.csv", "r")
table = list(csv.reader(f, delimiter=';'))
f.close()

#table vaut
#[ ['Nom', 'Note'], ['Toto', '14'], ['Titi', '13'], ['Tata', '18']]
```

`list(csv.reader(f,delimiter=c))` : crée un tableau de tableaux de chaînes de caractères, à partir du descripteur de fichier **f** en utilisant le délimiteur **c**. Si le délimiteur n'est pas donné, utilise la virgule.

`list(csv.DictReader(f,delimiter=c))` : crée un tableau de dictionnaires dont les clés sont les en-tête de colonnes.

Rappel sur les dictionnaires

Un dictionnaire (ou tableau associatif) associe des clés à des valeurs

```
mois = { "janvier" : 31, "février" : 28, …, "décembre" : 31 }
print(mois["janvier"]) # affiche 31
mois["février"] = 29   # mise à jour
del mois["novembre"]   # supprime l'entrée

for j in mois.values():
  print(j)  #affiche 31, 29, 31, …

for m in mois.keys():
  print(m)  #affiche janvier, février, …

for (m, j) in mois.items():
  print(m, "a", j, "jours") #affiche janvier a 31 jours, février a …
```

Chargement en dictionnaires


```
import csv
#on veut lire un fichier "fichier.csv" contenant
# Nom;Note
# Toto;14
# Titi;13
# Tata;18
f = open("fichier.csv", "r")
dic_table = list(csv.DictReader(f, delimiter=';'))
f.close()

#dic_table vaut
#[ { 'Nom':'Toto', 'Note' : '14'}, { 'Nom':'Titi', 'Note' : '13' }, … ]
```

Plan

1 Présentation du cours ✓
2 Le système Unix ✓
3 Le système Unix (2) ✓
4 Python (1) : expressions, types de bases, if/else ✓
5 Python (2) : boucles, tableaux, exceptions ✓
6 Python (3) : Textes, chaînes de caractères, entrées/sorties ✓
7 Python (4) : Fonctions ✓
8 Python (5) : Concepts avancés ✓
9 Applications (1) : Introduction aux réseaux ✓
10 Applications (2) : HTTP/HTTPS/HTML/CSS ✓
11 Traîtement de données
- 11.1 Lecture de fichiers de données ✓
- 11.2 Compréhensions de listes
- 11.3 Graphiques avec matplotlib

Souvenons nous ...

$$\{ x \in \mathbb{N} \mid x \bmod 2 = 0 \}$$

Notation mathématique *compacte* qui permet de définir l'ensemble des éléments *d'un autre ensmble* qui *vérifient une propriété*.

$$\{ (x,y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1 \}$$

« L'ensembles des points dont la norme vaut 1 » (i.e. le cercle de rayon 1)

C'est une notation très puissante, compacte et *déclarative* (elle exprime ce qu'on veut, pas comment on le calcule).

Tableaux en compréhension

Python permet de définir des tableaux en compréhension :

```
pairs = [ x for x in range(100) if x % 2 == 0 ]
#pairs vaut [ 0, 2, 4, 6, 8, … , 98 ]
```

La syntaxe (simplifiée) est :

```
[ e_r for x in e_l if c ]
```

où

- **e_l** doit être une expression renvoyant un tableau (directement, ou une fonction qui renvoie un tableau, ou **range**, ...)
- **e_r** est une expression résultat qui peut contenir la variable **x**
- **x** est une variable qui va prendre tour à tour les valeurs des éléments de **e_l**
- **c** est une expression booléenne (qui peut utiliser **x**) si elle est fausse, la valeur de **x** n'est pas utilisée et on passe à la suivante.

Tableaux en compréhension (2)

Python permet de définir des tableaux en compréhension :

Le code :

```
[ e_r for x in e_l if c ]
```

est équivalent à :

```
res = []
for x in e_l:
    if c:
        res = res + [e_r]
```

Exemple :

```
>>> [ (x-1, x+1) for x in range(10) if x % 3 = 0 ]
[(-1, 1), (2, 4), (5, 7), (8, 10)]
>>>
```

Utilité

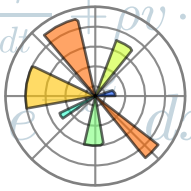
Un tableau donné par comprhénsion permet :

- de parcourir un ensemble de valeur
- de le filtrer pour garder des valeurs pertinentes
- de recombiner les données dans une liste résultante

C'est une opération tellement courante que les concepteurs de Python on jugé nécessaire de la mettre dans le langage.

Plan

1 Présentation du cours ✓
2 Le système Unix ✓
3 Le système Unix (2) ✓
4 Python (1) : expressions, types de bases, if/else ✓
5 Python (2) : boucles, tableaux, exceptions ✓
6 Python (3) : Textes, chaînes de caractères, entrées/sorties ✓
7 Python (4) : Fonctions ✓
8 Python (5) : Concepts avancés ✓
9 Applications (1) : Introduction aux réseaux ✓
10 Applications (2) : HTTP/HTTPS/HTML/CSS ✓
11 Traîtement de données
- 11.1 Lecture de fichiers de données ✓
- 11.2 Compréhensions de listes ✓
- 11.3 Graphiques avec matplotlib



matplotlib

Bibliothèque permettant de tracer des graphiques.

- Plusieurs types de graphiques (courbes, points, nuages de points, barres, camemberts, ...)
- Permet d'afficher dans une fenêtre graphique ou de sauver des fichiers d'images.
- Ne fait pas partie de la bibliothèque standard Python (doit être installé séparément)
- Open-Source
- Constitue le standard pour afficher des graphiques en Python

Le but est juste de faire une introduction ultra-rapide, la documentation est gigantesque, les possibilités d'utilisation très nombreuses.

Chargement de la bibliothèque


```
from matplotlib import pyplot
```

pyplot est lui-même un sous-module de **matplotlib**. On appelera toutes les fonctions en les préfixant par **pyplot.**

Il existe plusieurs façons d'utiliser la bibliothèque. On va voir la plus simple pour notre utilisation

Modèle de dessin

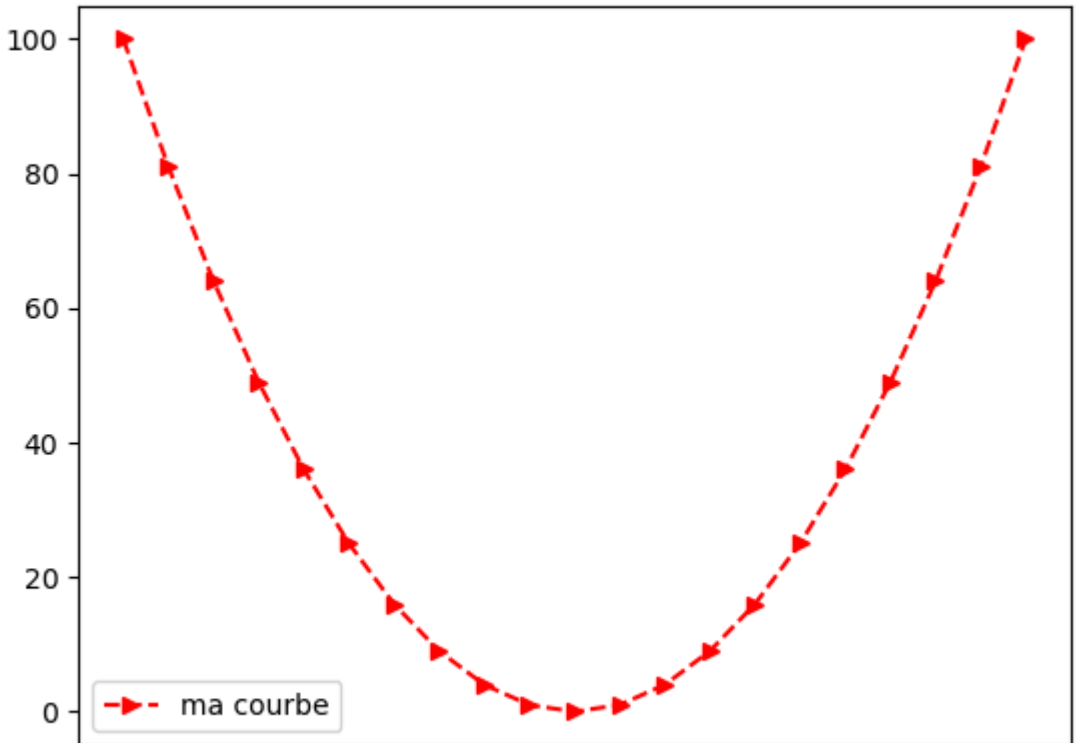
Il y a une notion de « graphique courant » qui est celui sur lequel on est en train de dessiner. Tant qu'on n'efface pas ce graphique, toutes les instructions de dessins vont s'ajouter les unes aux autres.
Forme générale du programme :

```
from matplotlib import pyplot

pyplot.plot(...)                #instructions de dessin
pyplot.grid()
pyplot.xlabel("temps")
pyplot.ylabel("energie")
pyplot.savefig("image1.png")  #sauvegarde dans un premier fichier
pyplot.clf()                    #on efface tout

...                             #autres instructions

pyplot.savefig("image2.png")  #sauvegarde dans un autre fichier
...
```

pyplot.plot

Graphique le plus basique. Trace des points (x,y) et les relie entre eux (ou pas)

```
tx = range(-10, 11)
ty = [ x * x for x in tabx ]
pyplot.plot(tx, ty, label='ma courbe', color='red', ls='--', marker='>')
pyplot.legend()
pyplot.savefig('g1.png')
```



pyplot.plot (2)

```
pyplot.plot(tx, ty, label=lab, color=col, ls=s, marker=m)
```

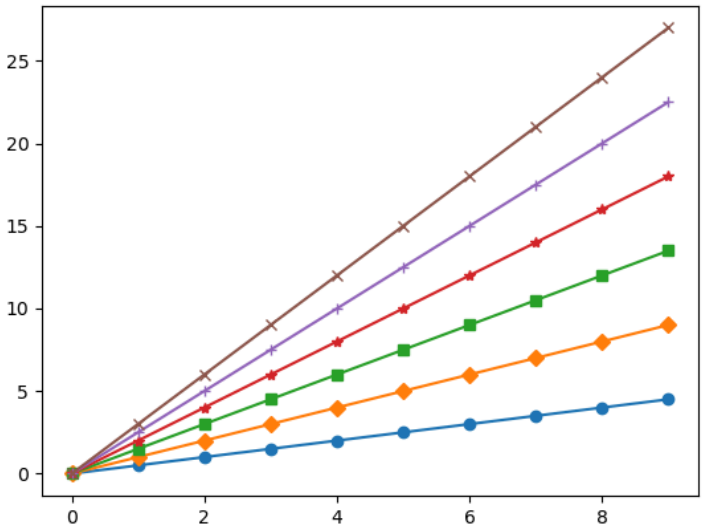
- **tx** est un tableau de nombres représentant les abscisses
- **ty** est un tableau de nombres représentant les ordonées. Il doit avoir la même taille que **tx**
- **lab** (optionnel) une chaîne de caractère représentant le nom de la coubre pour la légende.
- **col** (optionnel) la couleur de la courbe.
- **ls** (optionnel) le style des traits
- **m** (optionnel) le style des marqueurs

Les marqueurs

Il y a plein de styles de marqueurs. On en donne quelques uns:

symbole	**description**
'o'	cercle
'D'	diamant
's'	carré
'*'	étoile
'+'	plus
'x'	croix oblique
'<', '>', '^', 'v'	triangles orientés

```
tx = range(10)
pyplot.plot(tx, [x/2 for x in tx], marker='o')
pyplot.plot(tx, [x for x in tx], marker='D')
pyplot.plot(tx, [3*x/2 for x in tx], marker='s')
pyplot.plot(tx, [2*x for x in tx], marker='*')
pyplot.plot(tx, [5*x/2 for x in tx], marker='+')
pyplot.plot(tx, [3*x for x in tx], marker='x')
pyplot.savefig('g2.png')
```



Les couleurs

On peut donner les couleurs :

- par nom symbolique: 'red', 'blue', 'purple', ...
- en hexadécimal: `'#xxyyzz'`, avec `00 ≤ xx,yy,zz ≤ ff`
- en décimal avec transparence: `(x, y, z, a)`, avec `0 ≤ x,y,z,a ≤ 1`

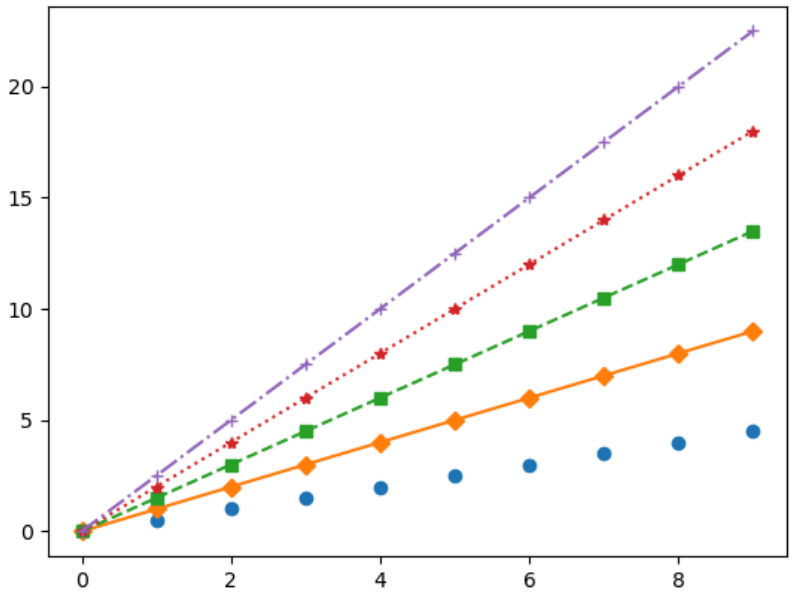
Si on mets plusieurs courbes sur un même dessin sans donner de couleurs, les couleurs sont choisies automatiquement.

Les styles de traits

le paramètre nommé **`ls`** (*linestyle*) peut prendre plusieurs valeurs :

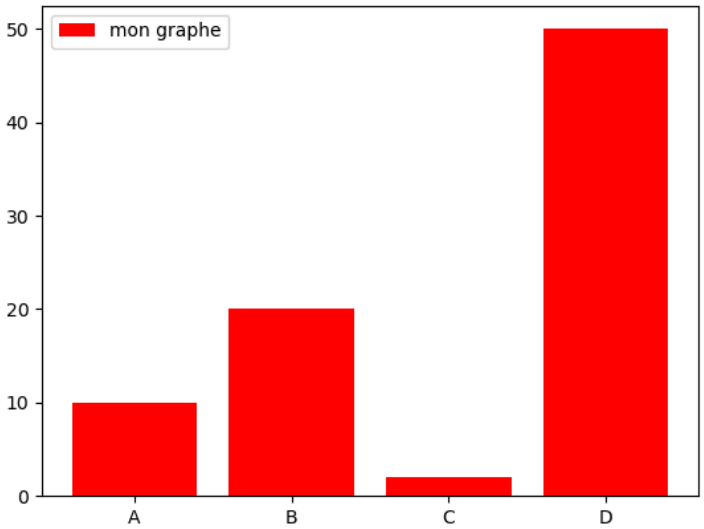
symbole	description
''	pas de trait
'-'	trait plein
'--'	tirets séparés
':'	pointillés
'-.'	tirets-points

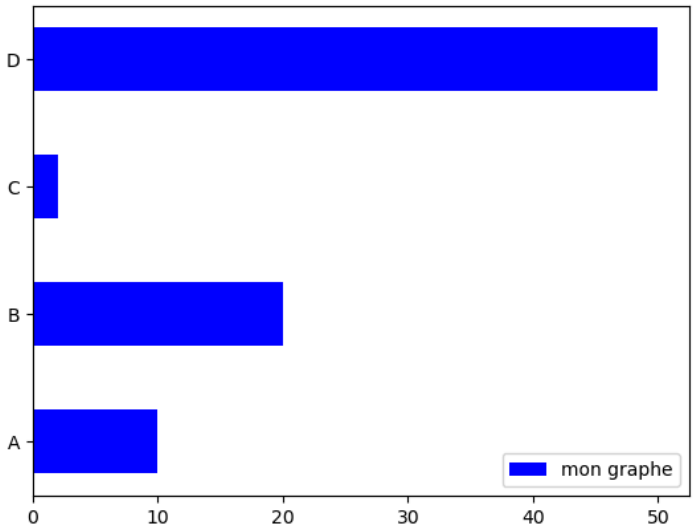
```
tx = range(10)
pyplot.plot(tx, [x/2 for x in tx], ls='', marker='o')
pyplot.plot(tx, [x for x in tx], ls='-', marker='D')
pyplot.plot(tx, [3*x/2 for x in tx], ls='--', marker='s')
pyplot.plot(tx, [2*x for x in tx], ls=':', marker='*')
pyplot.plot(tx, [5*x/2 for x in tx], ls='-.', marker='+')
pyplot.savefig('g3.png')
```



pyplot.bar et pyplot.barh

```
tx = [1, 2, 3, 4 ]
ty = [10, 20, 2, 50 ]
tl = ['A', 'B', 'C', 'D']
pyplot.bar(tx, ty, tick_label=tl, color='red', label='mon graphe')
pyplot.legend()
pyplot.savefig('g4.png')

pyplot.clf()
pyplot.barh(tx, ty, 0.5, tick_label=tl, color='blue', label='mon graphe')
pyplot.legend()
pyplot.savefig('g5.png')
```





pyplot.bar et pyplot.barh (2)

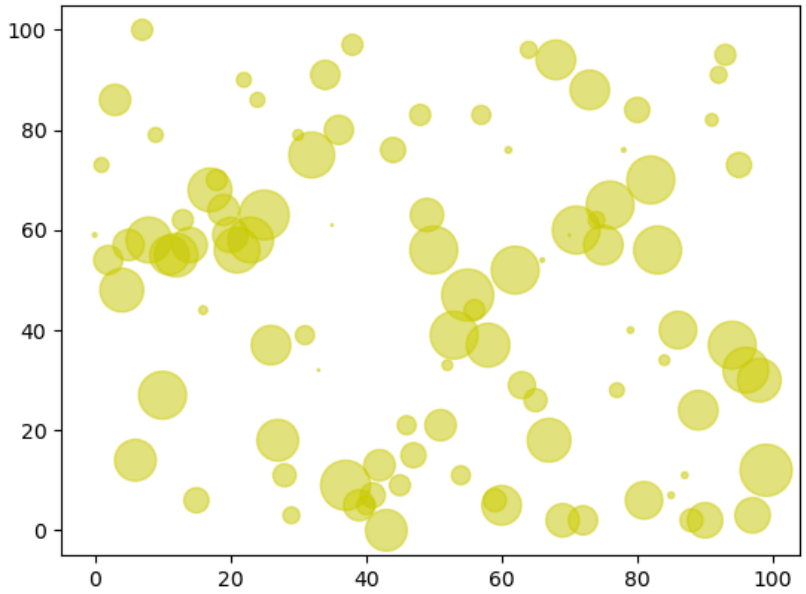
```
pyplot.bar(tx, ty, w, tick_label=tl, color=c, label=lab)
```

- **tx** est un tableau de nombres représentant les abscisses
- **ty** est un tableau de nombres représentant les hauteurs des barres. Il doit avoir la même taille que **tx**
- **w** (optionnel) est un flottant indiquant la largeur des barres
- **tl** est un tableau de chaînes représentant les étiquettes des barres. Il doit avoir la même taille que **tx**
- **lab** (optionnel) une chaîne de caractère représentant le nom de la coubre pour la légende.
- **col** (optionnel) la couleur de la courbe.

plot.barh trace les barres horizontalement

pyplot.scatter

```
from random import randint
tx = range(100)
ty = [ randint(0, 100) for x in tx ]
ta = [ randint(1, 15)**2 for x in tx ]
pyplot.scatter(tx, ty, s=ta, color=(0.8, 0.8, 0, 0.5))
pyplot.savefig('g6.png')
```



pyplot.scatter (2)

Nuage de points

```
pyplot.scatter(tx, ty, s=ta, color=col, label=lab,)
```

- **tx** est un tableau de nombres représentant les abscisses
- **ty** est un tableau de nombres représentant les ordonnées des points. Il doit avoir la même taille que **tx**
- **s** (optionnel) est un tableau indiquant l'aire de chaque point (par défaut constante). Il doit avoir la même taille que **tx**

Les autrs paramètres sont les même (**color=**, **legend=**)

Autres commandes

- **`pyplot.xlabel(s)`** : utilise la chaîne **`s`** comme étiquette pour l'axe d'es X
- **`pyplot.ylabel(s)`** : utilise la chaîne **`s`** comme étiquette pour l'axe d'es Y
- **`pyplot.legend()`** : ajoute une légende (il faut que les courbes aient été crées avec le paramètre nommé **`label=…`**)
- **`pyplot.grid()`** dessine une grille
- **`pyplot.clf()`** efface le dessin entièrement et supprime tous les graphes
- **`pyplot.savefig(f)`** sauve le dessin dans un fichier dont le nom est **`f`**. Le type d'image est déduit de l'extension (**`.png`**, **`.pdf`**, **`.svg`**, **`.jpg`**, ...)

```
from math import sin, cos
tx = [ x * 0.1 for x in range(-30,30) ]
ty1 = [ sin(x) for x in tx ]
ty2 = [ cos(x) for x in tx ]
pyplot.plot(tx, ty1, label='sin')
pyplot.plot(tx, ty2, label='cos')
pyplot.grid()
pyplot.legend()
pyplot.savefig('g7.png')
```