Introduction à Matplotlib et numpy pour l'affichage graphique

Nous allons utiliser d'abord le module pyplot de la bibliothèque matplotlib.
Pour l'importer, on trouve souvent dans la littérature l'alias « `plt` » : `import matplotlib.pyplot as plt`.
En mathématiques avec des élèves, je pense que c'est plus simple de tout importer d'un coup :

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

I. Afficher un point

On veut afficher un point de coordonnées (x, y) dans un repère. On utilise la fonction `plot` en lui indiquant l'abscisse et l'ordonnée puis la façon dont le point doit être représenté (une croix, un « + », un point etc).

Exemple : représentation du point (1,2) avec une croix :
`plot(1,2,"x")`
Attention! Une fois le graphique réalisé, il faut utiliser la fonction `show()` pour le rendre visible!

```
from matplotlib.pyplot import *

plot(1,2,"x")
show()
```

En annexe, vous trouverez les différentes façons de représenter un point (mais on les retrouve avec un `help(plot)`)

On peut choisir la couleur également en indiquant juste avant la représentation du point, la première lettre de la couleur souhaitée.

Exemple : représentation du point (1,2) avec une croix verte.
« g » comme « green ».
En annexe, vous trouverez les différents codes couleurs (idem, on les retrouve avec un `help(plot)`)

```
from matplotlib.pyplot import *

plot(1,2,"gx")
show()
```

On peut aussi modifier directement les paramètres `marker` et `color` de la fonction `plot` comme ceci :

```
from matplotlib.pyplot import *

plot(1,2,color="green",marker="x")
show()
```

(on peut intervertir et mettre `marker` en premier : `plot(1,2,marker="x", color="green")`.

Premières applications.
Supposons qu'une suite `u` soit définie sous forme d'une fonction Python (par exemple la suite (u_n) définie pour tout entier n par $u_n = n^2$).
On peut afficher ses différents termes graphiquement avec une boucle `for`.

Attention, il faut préciser la couleur car sinon Python va changer de couleur à chaque point. Ici, je précise `"kx"` (couleur black avec une croix).

```
from matplotlib.pyplot import *

def u(n):
    return n**2

def affiche(nb_termes):

    for k in range(nb_termes):
        plot(k,u(k),"kx")
    show()
```

Par exemple, ici affiche(5) permet d'afficher graphiquement les 5 premiers termes de la suite (u_n)

Le fait de pouvoir modifier la couleur permet pas mal d'applications (plusieurs exos sont proposés dans les différentes fiches). Par exemple, en échantillonnage, si on veut représenter différentes fréquences, on peut représenter d'une couleur celles qui appartiennent à l'intervalle de fluctuation et d'une autre, celles qui n'y sont pas.
Ou encore en géométrie, si on veut trouver un certain nombre de points vérifiant une condition, on peut représenter d'une couleur ceux qui vérifient la condition et d'une autre couleur ceux qui ne la vérifient pas etc …

II. Afficher plusieurs points

1. Stocker les abscisses et ordonnées dans des listes

On a vu qu'on pouvait afficher plusieurs points simplement en utilisant un `for`.
Mais selon le contexte, on peut avoir besoin de faire autrement : en général, pour afficher plusieurs points d'un coup, le mieux c'est de stocker dans deux listes toutes les abscisses et toutes les ordonnées. La syntaxe est alors la suivante : `plot(`*liste des abscisses*, *liste des ordonnées*`)`.
En plus, cette façon-là va permettre de relier les points entre eux (ce que ne permettent pas plusieurs plot isolés).

Exemple : on veut afficher et relier les points de coordonnées $(2;1)$, $(3;4)$, $(5;3)$.
On crée pour cela deux listes : une liste des abscisses contenant 2,3,5 et une liste des ordonnées contenant 1,4,3 puis on plot.
Les points sont reliés.
Bien sûr, on peut toujours modifier la couleur ainsi que la représentation du point.

```
from matplotlib.pyplot import *

abscisses = [2,3,5]
ordonnees = [1,4,3]
plot(abscisses,ordonnees)
show()
```

Si on veut quand même utiliser les listes mais que les points ne soient pas reliés, on utilisera plutôt `scatter` au lieu de `plot`.

```
from matplotlib.pyplot import *

abscisses = [2,3,5]
ordonnees = [1,4,3]
scatter(abscisses,ordonnees)
show()
```

Avec la fonction `scatter`, les points ne sont pas reliés.

NB : si on veut modifier la couleur ou la représentation du point avec `scatter`, il faut le faire en modifiant les paramètres `marker` et `color` obligatoirement.
Si on veut par exemple des croix rouges, ça donnera :

```
scatter(abscisses,ordonnees,color="red", marker="x")
```

tandis que ~~`scatter(abscisses,ordonnees,"rx")`~~ ne marchera pas (contrairement au plot !).

2. De nombreuses options pour changer l'apparence de la courbe

Si on relie avec un `plot`, on peut modifier l'affichage des différents segments qui relient en changeant le paramètre `linestyle`. Si on veut des pointillés :

```
plot(abscisses,ordonnees,linestyle="--")
```

D'autres paramètres sont ajustables, par exemple `linewidth` pour régler la largeur des segments.

character	description
`'-'`	solid line style
`'--'`	dashed line style
`'-.'`	dash-dot line style
`':'`	dotted line style

D'autres possibilités sont données avec un `help(plot)`

En tapant `help(plot)`, on accède à l'ensemble des paramètres ajustables à partir de là : (on peut s'amuser à tester pleins de choses)

```
  Here is a list of available `.Line2D` properties:

    agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
alpha: float (0.0 transparent through 1.0 opaque)
animated: bool
antialiased or aa: bool
clip_box: a `.Bbox` instance
clip_on: bool
clip_path: [(`~matplotlib.path.Path`, `.Transform`) | `.Patch` | None]
color or c: any matplotlib color
contains: a callable function
dash_capstyle: ['butt' | 'round' | 'projecting']
dash_joinstyle: ['miter' | 'round' | 'bevel']
dashes: sequence of on/off ink in points
drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']
figure: a `.Figure` instance
…
```

Application : Comment afficher par exemple la suite (u_n) définie pour tout entier naturel n par $u_n = n^2$ avec des listes ?

Pour la liste des abscisses, il suffit d'utiliser `range(0,nb_termes)`
Pour les ordonnées, on la crée en **compréhension.**
Python permet de créer des listes en s'inspirant de la notation des ensembles définis en compréhension en maths.
Ici, comme on ne veut pas relier, on utilise `scatter`.

```
from matplotlib.pyplot import *

def u(n):
    return n**2

def affiche(nb_termes):

    abscisses = range(0,nb_termes)
    ordonnees = [u(n) for n in abscisses]
    scatter(abscisses,ordonnees,marker='x')
    show()
```

Remarque : pour créer la liste des ordonnées, la syntaxe `u(abscisses)` ne fonctionne pas. Par contre, nous verrons que si `abscisses` représente un tableau issu de `numpy`, alors on pourra procéder ainsi (comme d'ailleurs avec *Matlab*).

III. Afficher la courbe représentative d'une fonction

Problème : on veut par exemple afficher sur l'intervalle $[0;10]$ la courbe de la fonction f définie pour tout x réel par $f(x) = x^2$.

Il faut d'abord créer notre fonction en Python.

```
def f(x):
    return x**2
```

Pour la tracer, l'astuce est de relier beaucoup de points de la courbe, très proches les uns des autres.
Il faut alors subdiviser l'intervalle $[0;10]$. Pour cela, on utilise des fonctions de `numpy`. Il faut alors l'importer :

```
from numpy import *
```

Tester la commande suivante `ma_subdvision = linspace(0,10)`. On obtient une subdivision de l'intervalle $[0;10]$ avec 50 points.
Cette fonction retourne un tableau (type `numpy.ndarray`, ça ressemble fortement à une liste).

On peut changer le nombre de points de la subdivision en renseignant un troisième paramètre. Si on veut par exemple une subdivision à 100 points, on utilisera `linspace(0,10,100)`.

Pour tracer, il suffit de stocker dans une liste `abscisses` les points de notre subdivision et dans une liste `ordonnees` les images de ces points par la fonction f

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

def f(x):
    return x**2

def affiche():
    abscisses = linspace(0,10,100)
    ordonnees = [f(x) for x in abscisses]  # remarque : ordonnees = f(abscisses) fonction ici car abscisses est un tableau de numpy et non une liste
    plot(abscisses,ordonnees)
    show()
```

.

Remarque : pour définir `ordonnees`, on peut écrire `ordonnees = f(abscisses)`. Python comprend alors que cela signifie « calcule moi l'image de tous les éléments de ta liste abscisses ». Mais il faut bien savoir que ça marche car `abscisses` est un tableau de `numpy`[1]. Si `abscisses` est de type `list`, ça ne fonctionne pas.
Si on veut régler le pas plutôt que le nombre de points, on peut aussi utiliser la fonction `arange` de `numpy`.

[1] `numpy` s'inspire de Matlab où l'on procède ainsi.

Par exemple, si on veut une subdivision de $[0;10]$ avec un pas de 0.5, on utilise :

```
arange(0,10,0.5)
```

IV. Réglage des axes et de la fenêtre graphique

Pour avoir un repère orthonormé	On rajoute la commande `axis("equal")`
Pour avoir deux droites graduées qui se coupent en O et non quatre droites graduées autour.	On peut utiliser la fonction `axesnormaux()` qui se situe dans un module que j'ai nommé `utile` (on peut changer le nom). Du coup, il faudra importer le module `utile` dans vos scripts : `from utile import *`

Si on veut régler la fenêtre graphique comme suit (a, b, c, d sont des nombres de votre choix) :
$\begin{cases} x_{\min} = a \\ x_{\max} = b \\ y_{\min} = c \\ y_{\max} = d \end{cases}$
	On utilise `xlim` et `ylim` :
	`xlim(a,b)`
	`ylim(c,d)`

Si on veut afficher que certaines graduations en abscisses et/ou en ordonnées.	On utilise les fonctions `xticks` et `yticks`. On leur passe en paramètre les listes des graduations que l'on souhaite afficher.
	Si on veut par exemple afficher sur l'intervalle $[0;10]$ uniquement les graduations 0,5,10, alors :
	`xticks([0,5,10]).`
	Idem en ordonnées.
	
	En particulier, si on ne veut aucune graduation en abscisse par exemple, on utilise `xticks([]).`
	
	On peut même aller plus loin. Si on veut afficher uniquement les graduations 0,5,10 mais qu'on veut leur donner une étiquette en les nommant autrement que 0,5,10 (par exemple debut, milieu, fin),il faut mettre la liste des étiquettes dans `xticks` :
	`xticks([0,5,10], ["debut","milieu","fin"])`

Si on veut ajouter une légende en abscisses ou en ordonnées.	En abscisses : `xlabel(`*texte*`) et ylabel(`*texte*`)`
Si on veut ajouter un titre à notre graphique.	`title(`*texte*`)`. On peut même mettre du LaTeX!!
Si on veut ajouter une grille.	`grid()`

Si on veut ajouter une légende à un tracé. Par exemple, on trace la courbe de la fonction `cosinus` et on veut ajouter une légende qui dit que la courbe correspond à `"cosinus"`	Dans notre `plot`, on rajoute `label="cosinus"` :
	`plot(…,…, label = "cosinus")`
	Puis on ajoute l'instruction `legend()`. On peut renseigner un paramètre `loc` qui indique la position de la légende (avec un `help(legend)`, on en sait plus).
	Par exemple, en haut à droite donnerait :
	`legend("upper right")`

V. Bilan : créer une fonction `tracer` qui permet de tracer une fonction passée en paramètre.

Une fonction `tracer(f,depart,fin,pas)` qui prend en paramètre une fonction `f` et trois nombres réels et trace la fonction `f` dans l'intervalle `[depart,arrivee]` en utilisant un pas renseigné en paramètre.

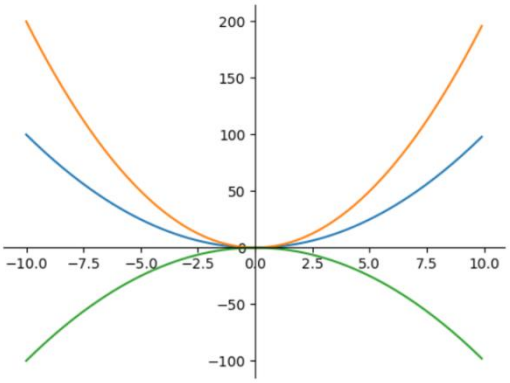
```
def tracer(f,depart,arrivee,pas):

    abscisses = arange(depart,arrivee,pas)
    ordonnees = f(abscisses)
    axesnormaux()
    plot(abscisses,ordonnees)
```

Il est mieux de ne pas mettre le `show()` à l'intérieur de la fonction mais plutôt dans la console après l'appel de `tracer`. En effet, si on veut tracer plusieurs fonctions dans le même graphique, il suffit d'appeler plusieurs fois `tracer` et de mettre à la fin `show()`.
Par exemple, on veut tracer sur $[-10,10]$ les fonctions f, g, h définies pour tout x réel par $f(x) = x^2, g(x) = 2x^2, h(x) = -x^2$.
On les définit d'abord en Python :

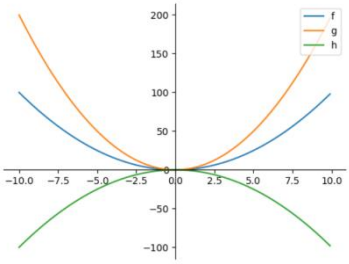
```
def f(x):
    return x**2

def g(x):
    return 2*x**2

def h(x):
    return -1*x**2
```

Puis on appelle la fonction `tracer` trois fois et on se fait un `show()`.

```
tracer(f,-10,10,0.1)
tracer(g,-10,10,0.1)
tracer(h,-10,10,0.1)
show()
```



Remarque : on peut éventuellement vouloir afficher une légende indiquant quelle fonction correspond à quelle courbe comme le montre la figure suivante. J'ai bidouillé un truc qui le fait (voir fichier python joint, fonction `tracer_avec_legende`) mais c'est pas super joli…

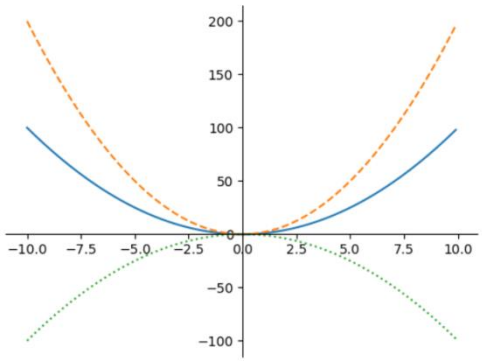


Autre remarque : Ou alors, dans notre fonction `tracer`, on peut ajouter un paramètre qui demande le style de la courbe à afficher dans notre fonction (en pointillé, trait plein etc ..)

```
def tracer2(f,depart,arrivee,pas,style_ligne="-"):

    abscisses = arange(depart,arrivee,pas)
    ordonnees = f(abscisses)
    axesnormaux()
    plot(abscisses,ordonnees,linestyle=style_ligne)
```

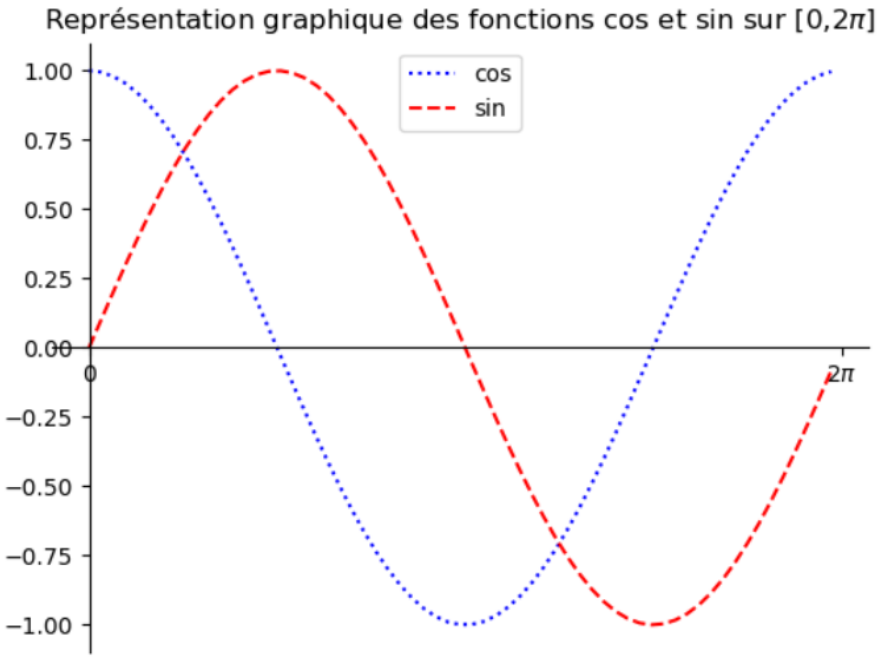
Le fait d'écrire `style_ligne="-"` permet de partir sur ce réglage par défaut (si l'utilisateur ne renseigne pas ce paramètre, aucune erreur n'est affichée, c'est ce paramètre qui sera pris en compte).

```
tracer2(f,-10,10,0.1)
tracer2(g,-10,10,0.1,style_ligne="--")
tracer2(h,-10,10,0.1,style_ligne=":")
show()
```



VI. Exercice bilan

Essayez de faire l'exercice suivant qui utilise pas mal d'options introduites précédemment.
Ecrivez un script permettant d'afficher la figure suivante (si vous ne connaissez pas bien LaTeX, on peut aussi se contenter d'afficher "pi" à la place de "π").
(le code est disponible dans le fichier python joint avec la fonction `affiche_cos_sin`).

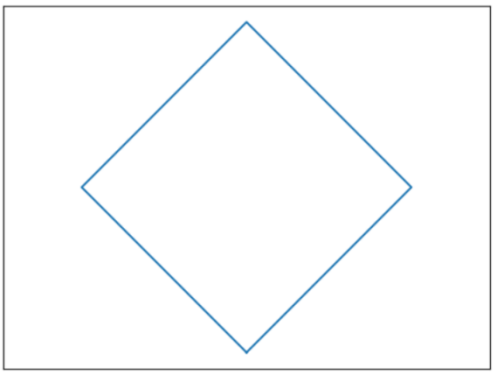


VII. Affichage de formes géométriques

On peut aussi afficher des formes géométriques en reliant des points. Il sera nécessaire de travailler dans un repère orthonormé (instruction `axis("equal")`) afin que les figures ne soient pas déformées.

Par exemple un carré : on relie les points $(0,-1),(1,0),(0,1),(-1,0)$ et $(-1,0)$ (le dernier point doit être relié au point de départ, sinon ça ne fera qu'un triangle).
On peut enlever les axes ici avec `xticks([])` et `yticks([])`.

```
def affiche_carre():

    abscisses = [0,1,0,-1,0]
    ordonnees = [-1,0,1,0,-1]
    plot(abscisses,ordonnees)
    axis("equal")
    xticks([])
    yticks([])
    show()
```

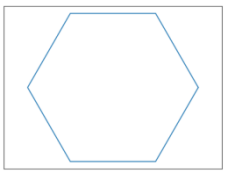


De façon générale, si on veut afficher un polygone régulier à n côtés, il faut relier les points $\left(\cos\left(\frac{2k\pi}{n}\right),\sin\left(\frac{2k\pi}{n}\right)\right)$ pour $k=0,1,\dots,n-1$.

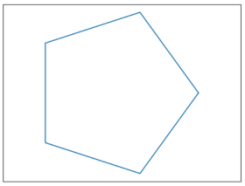
Il devient alors assez simple de faire une fonction `affiche_polygone(n)` permettant d'afficher un polygone à n côtés. Il suffit de créer la liste des abscisses, c'est l'ensemble $\left\{\cos\left(\frac{2k\pi}{n}\right), k\in\{0,1,\dots,n-1\}\right\}$ (la liste est créée en compréhension avec une syntaxe très proche de cet ensemble). Egalement, la liste des ordonnées, c'est l'ensemble $\left\{\sin\left(\frac{2k\pi}{n}\right), k\in\{0,1,\dots,n-1\}\right\}$ et on plot tout ça. Attention tout de même : le dernier point doit être relié au premier pour fermer la figure. Ce premier point est $\left(\cos\left(\frac{2\times0\times\pi}{n}\right),\sin\left(\frac{2\times0\times\pi}{n}\right)\right)$, soit le point $(1,0)$.

Le code source est le suivant.

```
def affiche_polygone(n):

    abscisses = [cos(2*k*pi/n) for k in range(n)] + [1]
    ordonnees = [sin(2*k*pi/n) for k in range(n)] + [0]
    axis("equal")
    xticks([])
    yticks([])
    plot(abscisses,ordonnees)
    show()
```

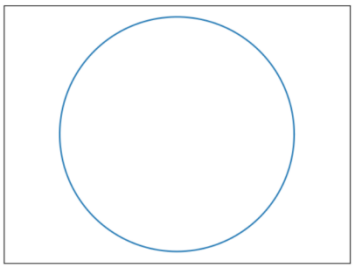


`affiche_polygone(6)`



`affiche_polygone(5)`

Pour afficher un cercle, il suffit de prendre n assez grand.



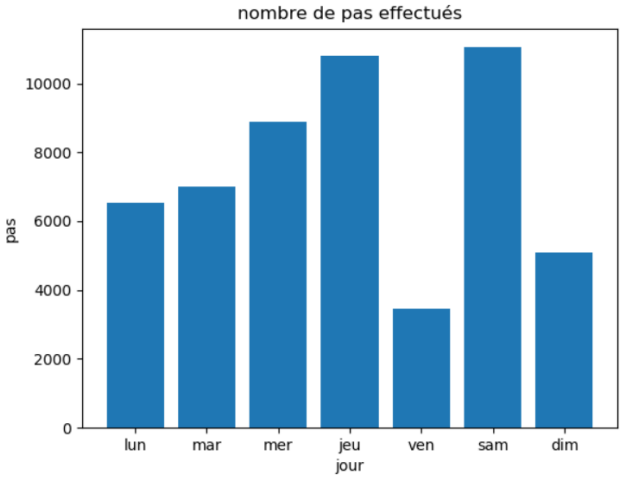
`affiche_polygone(10000)`

VIII. Histogrammes en statistiques

La fonction principale sera la fonction `bar`.
Partons d'un exemple. Une personne compte ses pas quotidiens pendant 1 semaine. Les résultats sont donnés dans le tableau ci-dessous.

Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
6534	7000	8900	10786	3467	11045	5095

On veut représenter l'histogramme ci-contre.
Plusieurs éléments dans cette figure :



- Le titre « nombre de pas effectués » s'effectue grâce à la fonction `title`.
- De même que « jour » et « pas » en abscisses et ordonnées se fait avec `xlabel` et `ylabel`.

Ensuite, c'est la fonction `bar` va permettre d'afficher l'histogramme.
Elle marche comme `plot` quasiment.
Elle prend deux listes : la liste des abscisses et la liste des ordonnées. Pour chaque point $(x;y)$ (où x appartient à la liste `abscisses` et `y` à la liste `ordonnees`), elle va tracer un rectangle de hauteur y et d'une certaine largeur au niveau de x (on rajoute le paramètre `align="center"` pour que le rectangle soit centré autour de x).

Il suffit ensuite avec un `xticks` de changer les étiquettes des graduations en mettant lun, mar, mer etc…

Voici le code.

```
mespas = [6534,7000,8900,10786,3467,11045,5095]
positions = [1,2,3,4,5,6,7]
lesjours = ['lun','mar','mer','jeu','ven','sam','dim']
bar(positions,mespas,align="center")
xticks(positions,lesjours)
xlabel('jour')
ylabel('pas')
title('nombre de pas effectués')
show()
```

On peut faire une fonction pour automatiser le truc..

```
def histogramme(donnees, etiquettes):
    nombre_barres = len(donnees)
    positions = range(1,nombre_barres+1)
    bar(positions, donnees, align='center')
    xticks(positions, etiquettes)
    ylabel('pas')
    xlabel('jour')
    title('nombre de pas effectués')
    show()
```

Puis on l'appelle :

```
mespas = [6534,7000,8900,10786,3467,11045,5095]
lesjours = ['lun','mar','mer','jeu','ven','sam','dim']
histogramme(mespas,lesjours)
```

IX. Annexe

Les différentes façons de représenter un point avec `plot`.

`'o'` : rond.
`'s'` : carré (square).
`'+'` : croix en forme de +.
`'x'` : croix en forme de x.
`'*'` : étoile.
`'D'` : losange (diamond).
`'d'` : losange allongé.
`'H'` : hexagone ('h' est aussi un hexagone, mais tourné).
`'p'` : pentagone.
`'.'` : point.
`'>'` : triangle vers la droite (`'<'` pour vers la gauche).
`'v'` : triangle vers le bas (`'^'` pour vers la haut).
`'|'` : trait vertical (`'_'` pour trait horizontal).
`'1'` : croix à 3 branches vers le bas ('2' vers le haut, '3' vers la gauche, '4' vers la droite).

Les différents codes couleur.

Code	Couleur en anglais	Couleur en français
b	blue	bleu
g	green	vert
r	red	rouge
c	cyan	cyan

Code	Couleur en anglais	Couleur en français
m	magenta	magenta
y	yellow	jaune
k	black	noir
w	white	blanc

Pour aller plus loin :

http://www.python-simple.com/python-matplotlib/graphes-multiples.php