

The background features a gradient from dark purple at the top to deep blue at the bottom, speckled with white dots resembling stars. Overlaid on this are several faint, white, semi-transparent circular and arc-like patterns. Some of these patterns include tick marks and numbers, suggesting a circular scale or a polar coordinate system. The numbers visible include 40, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, and 260. The overall aesthetic is technical and scientific.

# TRACER DES GRAPHES AVEC MATPLOTLIB

Vous voulez tracez de somptueux graphes pour épater la galerie ? En voulez-vous ? En voilà !

Avant de commencer à vous expliquer comment cela fonctionne, sachez qu'il existe plusieurs types de graphes. Dans ces magnifiques diapositives delamortquituent, 3 d'entre eux seront abordés : les courbes, les diagrammes en barres (ou bâtons) et les histogrammes.

Vous allez donc apprendre à tracer des graphes ! Mais seulement en 2D. Si vous voulez faire de la 3D, libre à vous de vous documenter.

# SOMMAIRE

• Généralités .....	6
• Courbes .....	12
• Compléments additionnels .....	48
• Courbes et fonctions mathématiques .....	88
• Diagrammes en barres .....	91
• Histogrammes .....	128
• Diagramme circulaire .....	151
• Subplot (fenêtre multi-graphes) .....	169
• Lien avec Numpy .....	176
• Exercices .....	178

# AVANT DE COMMENCER

- Assurez-vous que le module matplotlib soit installé sur votre machine. Attention ! « mat » et non pas « math » !
- Importez-le au début de votre script. Attention ! « mat » et non pas « math » !

```
import matplotlib
```

- Matplotlib étant un module contenant des « sous-modules », un de ces sous-modules nous servira principalement à tout faire. Les autres modules ne sont là que pour compléter. Donc le sous-module qui nous intéresse ici est le suivant : « pyplot ». Importez le comme suit :

```
import matplotlib.pyplot
```

- Afin d'éviter de taper « matplotlib.pyplot » à chaque appel de fonction de ce module, celui-ci est souvent abrégé par « plt ». Mais libre à vous de choisir n'importe quel mot (ou de tout retaper à chaque fois).

```
import matplotlib.pyplot as plt
import matplotlib.pyplot as coucou
import matplotlib.pyplot as OMG
import matplotlib.pyplot as burgerking
```



Merci de votre patience et  
de votre attention ! Nous  
allons pouvoir commencer  
avec ...

## ... QUELQUES GÉNÉRALITÉS !

Domage ! Vous allez attendre encore un peu avant de tracer de belles lignes et des barres !

Mais qu'est-ce qu'un graphe ?

- Une/des courbe(s), une/des barre(s), un/des dessin(s)
- Avec un/des axe(s) gradué(s) ou non
- Et une/des légende(s) !

Oui ça fait beaucoup, mais voyons les un par un, dans le désordre ...

## EN COMMENÇANT PAR LES AXES !

Lorsque vous placerez un point sur le graphe, vos axes et graduation seront créés automatiquement. Il n'y a pas vraiment de commande spéciale à exécuter. Les axes s'adaptent aux coordonnées des points que vous avez placé. Vous pourrez les modifier par la suite.

Lorsque que vous créez vos graphes, vous les faites « à l'aveugle ». En effet, il faut imaginer comment votre graphe est et sera constitué. Il n'y a pas de fenêtre ouverte en permanence qui vous permet de suivre l'avancement de la création de votre graphe.

Lorsque vous ajoutez des modifications, il faudra à chaque fois lancer votre programme pour voir ce que vous avez ajouté/supprimé/changé. C'est chiant, mais c'est comme ça.

Si votre programme doit tourner 5 minutes avant d'afficher le graphe désiré et qu'il ne vous convient pas, ne vous embêtez pas. Essayez d'abord avec un petit nombre de données, puis ajouter les modifications nécessaires. Enfin lancez à nouveau sur vos données principales.



# AFFICHER LE GRAPHE

Après avoir bien placé vos points, fait vos modifications, il est grand temps de voir le résultat !

Pour cela il suffit d'afficher le graphe dans une fenêtre avec la commande suivante :

**`plt.show()`**

N'oubliez pas le nom du module de la fonction avant le nom de la fonction.

Vous pouvez essayer de taper cette ligne dans un éditeur de texte puis de lancer votre script mais rien ne s'affichera. En effet, vous n'avez pas encore placé de point ni créé votre graphe, mais lorsque que vous l'aurez fait, il faudra taper cette commande pour afficher ce que vous avez fait.

# FENÊTRE DU GRAPHE

Votre graphe sera affiché dans une nouvelle fenêtre. Il existe une commande qui permet de créer cette fenêtre. Il s'agit de la suivante : **plt.figure()**.

En fait vous n'en avez pas vraiment besoin puisqu'en plaçant vos points sur le graphe et en l'affichant, la fenêtre est automatiquement créée.

Vous pouvez passer une chaîne de caractères entre les ( ) de la commande pour donner un titre à la fenêtre, mais il n'y a pas vraiment d'utilité.

Vous pouvez tester rapidement, dans votre script avec **plt.figure(« blablabla »)** suivi d'un **plt.show()**, pour voir à quoi s'attendre.

Dans mon cas, je n'ai jamais eu à utiliser cette commande, donc libre à vous de l'utiliser ou non.

## FERMETURE « PROPRE »

Lorsque vous afficherez plusieurs graphes à la suite, ceux-ci sont gardés en mémoire. Au fur et à mesure que les graphes s'enchaîneront, la mémoire sera saturée. Afin d'éviter cela, il suffit de fermer chaque fenêtre de vos graphes « proprement » avec la commande suivante :

**plt.close()**

Vous devez bien sur fermer manuellement la fenêtre avec un click de souris, mais les données de votre graphe ne seront pas gardés en mémoire, et vous n'aurez pas de problème pour afficher les suivantes !

À placer donc à la fin de toutes vos commandes graphiques de chaque graphe.

Si vous n'avez qu'un seul graphe ou deux à afficher, il n'est pas nécessaire d'utiliser **plt.close()**.

# LES COURBES ! (ENFIN !)

C'est quoi une courbe ?

C'est une ligne, courbée me direz-vous. Et bien moi, je dirais même plus que ce sont des points !

En effet, lorsque vous allez tracer une courbe, vous allez placer des points sur votre graphe et les relier entre eux par des lignes si vous le désirez.



# PLACER UN POINT

Un point est défini par des coordonnées. En 2D, deux coordonnées donc : x et y.

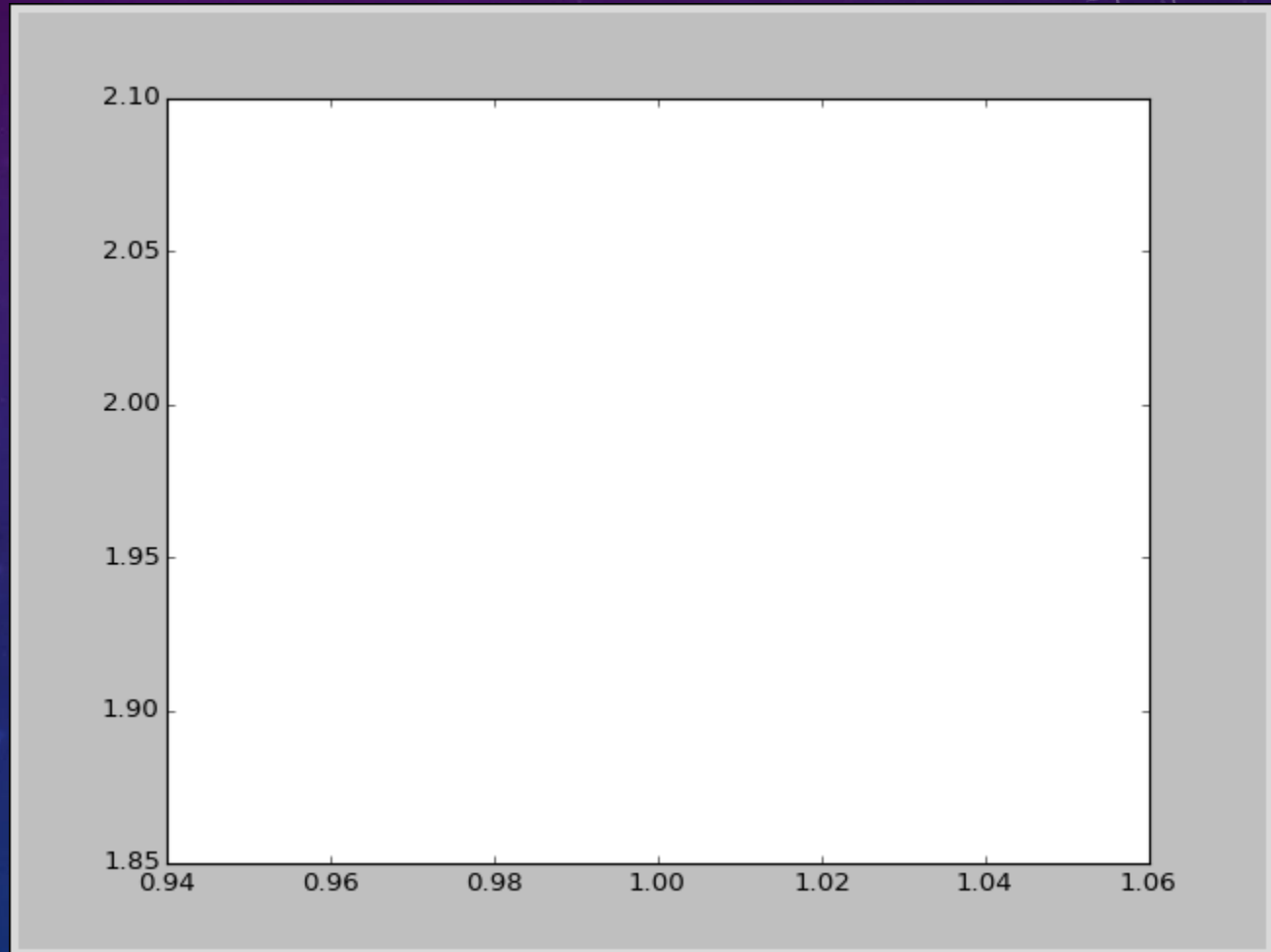
Pour placer un point de coordonnées **(x, y)** sur votre graphe (imaginaire pour l'instant), la syntaxe est la suivante :

**`plt.plot(x, y)`**

Essayez avec un x et y que vous choisissiez puis affichez le résultat.

# EXAMPLE

```
plt.plot(1, 2)  
plt.show()
```



Il n'y a rien n'est-ce pas ?

Pas vraiment. Votre point est bien placé, mais il n'est pas visible. Remarquez que vos axes sont plus ou moins centrés en ce point.

Afin de faire afficher votre point, il est nécessaire de préciser sa forme en argument de la fonction **plt.plot()**, entre guillemets « », et ce avec ou sans le mot clé **marker** avant, après les coordonnées du point.

Pour la forme d'un point, rien de plus simple qu'un point « . ». Sachez qu'il existe d'autres formes possibles, mais pour le moment, « . » suffit pour exemple. Je/vous vais/pouvez également utiliser « o », qui est également un point, mais plus gros, pour plus de lisibilité dans les exemples qui suivent.

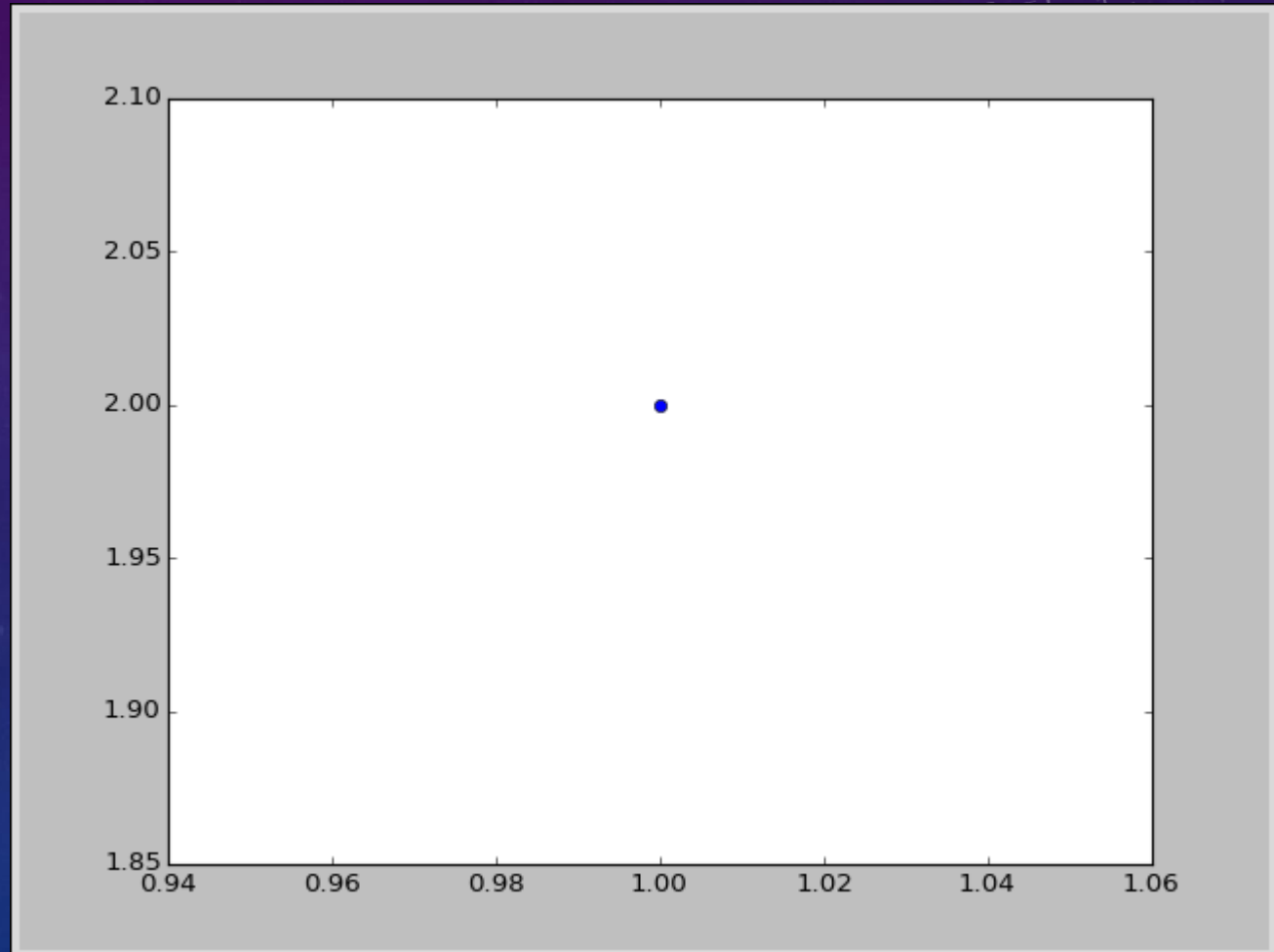
Essayez donc !

# EXEMPLE

```
plt.plot(1, 2, "o")  
plt.show()
```

Ou bien

```
plt.plot(1, 2, marker="o")  
plt.show()
```





## PLACER UN AUTRE POINT

Vous me direz que pour tracer une courbe (ou une droite), il faut au moins deux points, et bien il suffit d'en placer un autre !

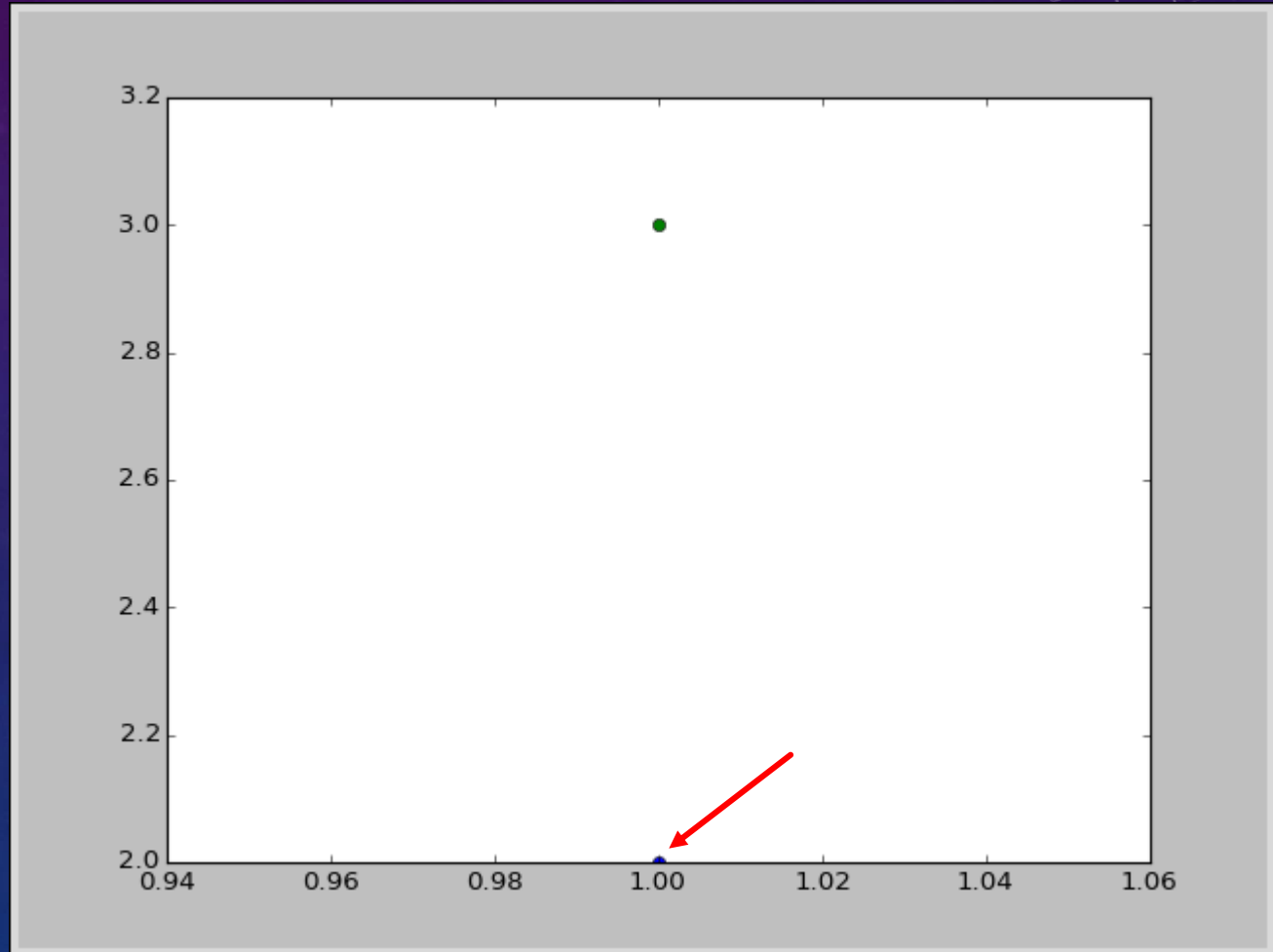
Pour cela, répéter la commande précédente, avec de nouvelles coordonnées.

# EXEMPLE

```
plt.plot(1, 2, "o")  
plt.plot(1, 3, "o")  
plt.show()
```

Ou bien

```
plt.plot(1, 2, marker="o")  
plt.plot(1, 3, marker="o")  
plt.show()
```



## ET LA LIGNE QUI RELIE LES DEUX POINTS ?

Patience ! Pas tout de suite !

Vous avez vu que pour placer plusieurs points il faut répéter la commande **plt.plot()**. Lorsque vous avez une série de données, contenues dans une/des listes par exemple, le réflexe serait d'appeler la commande autant de fois qu'il y a de points à placer.

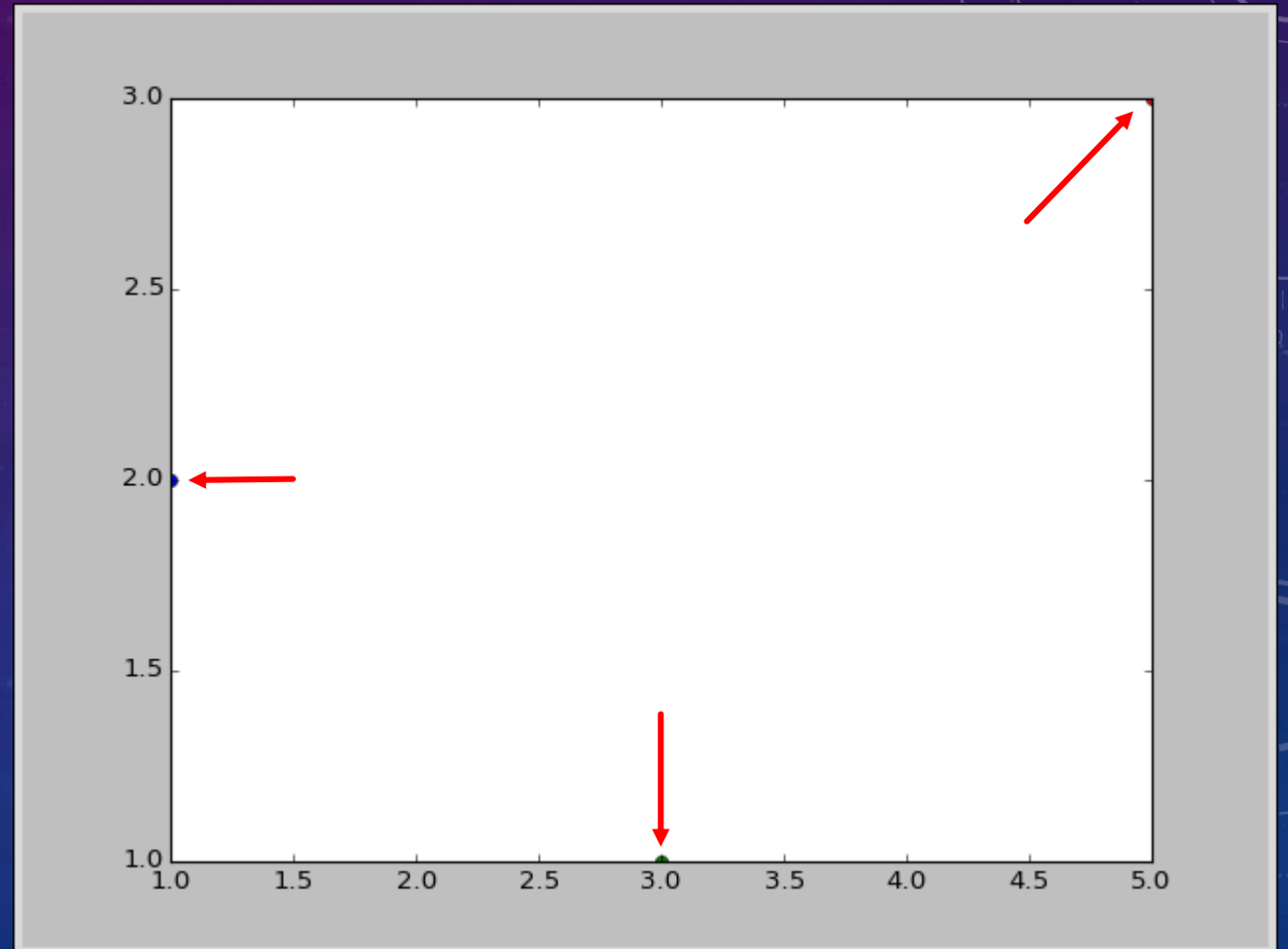
Au lieu de taper 36000 lignes de **plt.plot()**, une boucle for qui parcourrait la liste de données serait plus efficace (on pourrait utiliser while, mais for est plus adapté).

# EXEMPLE

```
points = [(1,2), (3,1), (5,3)]  
  
plt.plot(points[0][0], points[0][1], "o")  
plt.plot(points[1][0], points[1][1], "o")  
plt.plot(points[2][0], points[2][1], "o")  
  
plt.show()
```

Ou bien

```
points = [(1,2), (3,1), (5,3)]  
  
for p in points :  
    plt.plot(p[0], p[1], "o")  
  
plt.show()
```





# LA FAMEUSE LIGNE/COURBE !

Pour relier 2 points de coordonnées **(x1 ,y1)** et **(x2, y2)**, il suffit d'utiliser la commande suivante :

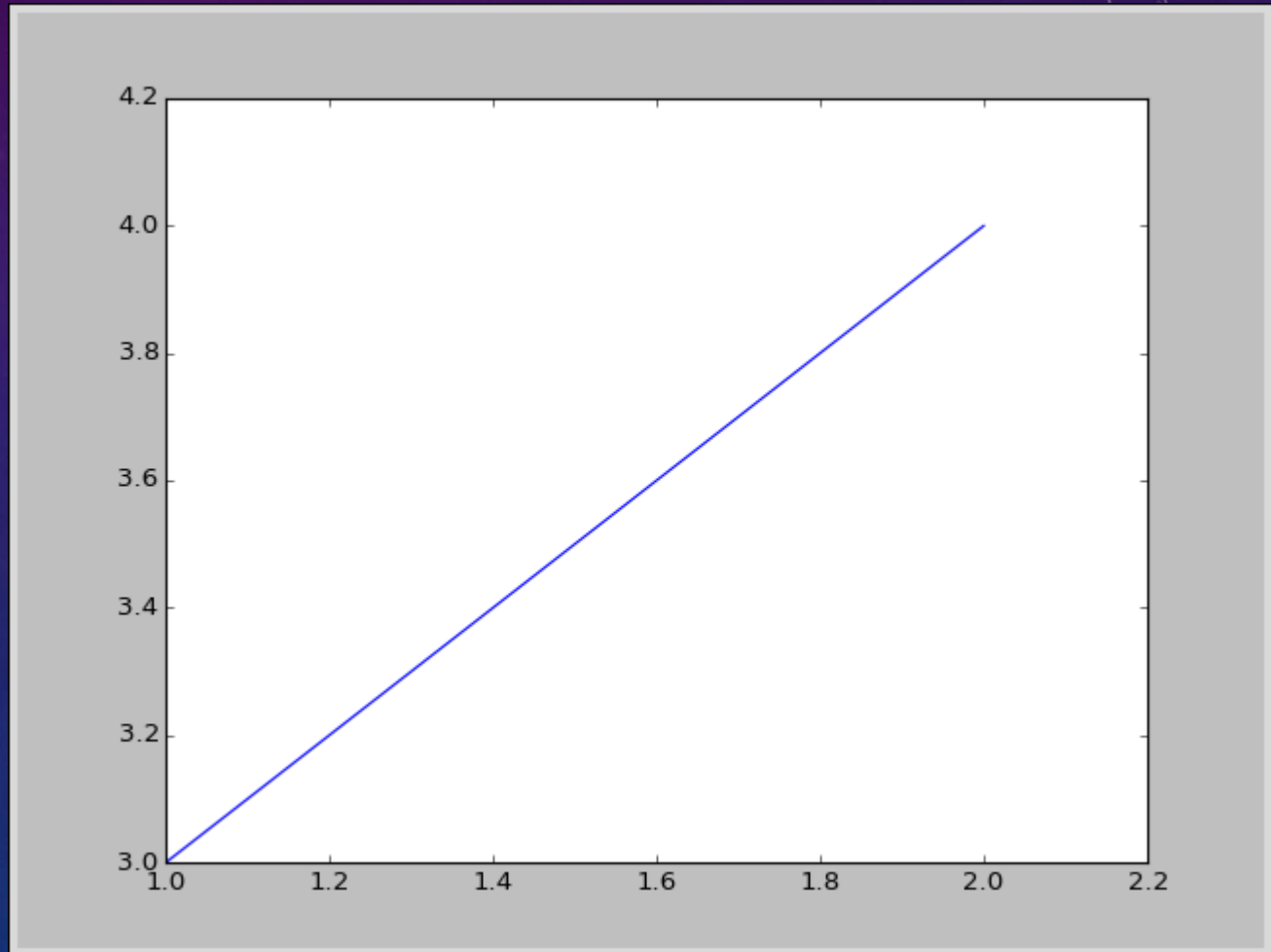
**`plt.plot([x1, x2], [y1, y2])`**

Explication : vous passez en arguments de la fonction `plot()`, 2 listes : une pour les coordonnées x de vos points, et une autre pour les y. C'est de cette façon que vous allez pouvoir relier plusieurs points et ainsi tracer votre courbe.

Par défaut, la ligne tracée est une ligne pleine. Nous verrons par la suite qu'il est possible de modifier cela.

# EXAMPLE

```
plt.plot([1,2], [3,4])  
plt.show()
```



## POUR PLUSIEURS POINTS (3+)

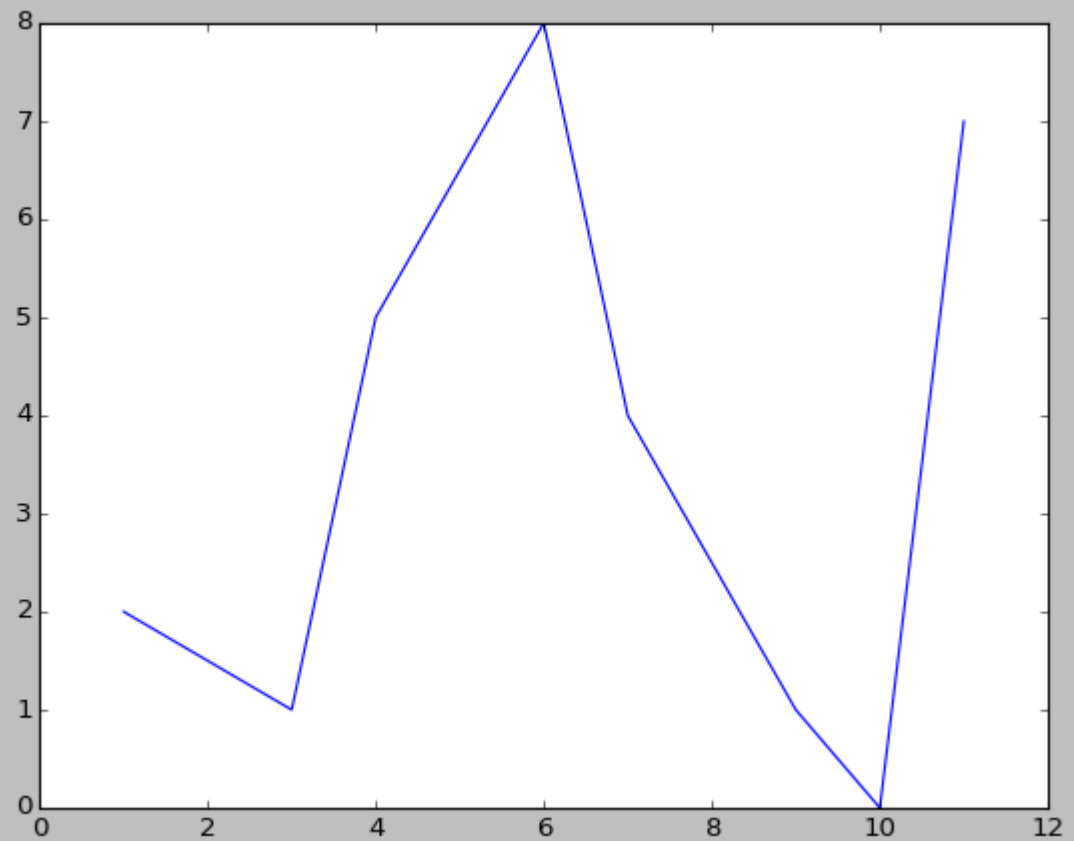
Lorsque vous passez la liste des x et la liste des y en arguments de la fonction `plot()`, faites attention à ce que les deux listes soient de même dimension. Sinon vous aurez une erreur.

Pour tracer plusieurs courbes sur le même graphe, il suffit de répéter **`plt.plot()`** avec en arguments vos séries de données, autant de fois que vous souhaitez avoir de courbes. Vous pouvez également le faire en une ligne, mais ce n'est pas vraiment conseillé si vous avez beaucoup.

Note : lorsque vous affichez plusieurs courbes sur un même graphe, si vous ne précisez pas la couleur de chaque courbe, une couleur leur est automatiquement attribuée. Nous verrons comment modifier la couleur plus loin. Même chose pour des points, s'ils sont affichés avec des `plot()` différents.

# EXAMPLES

```
x = [1, 3, 4, 6, 7, 9, 10, 11]  
y = [2, 1, 5, 8, 4, 1, 0, 7]  
  
plt.plot(x, y)  
plt.show()
```

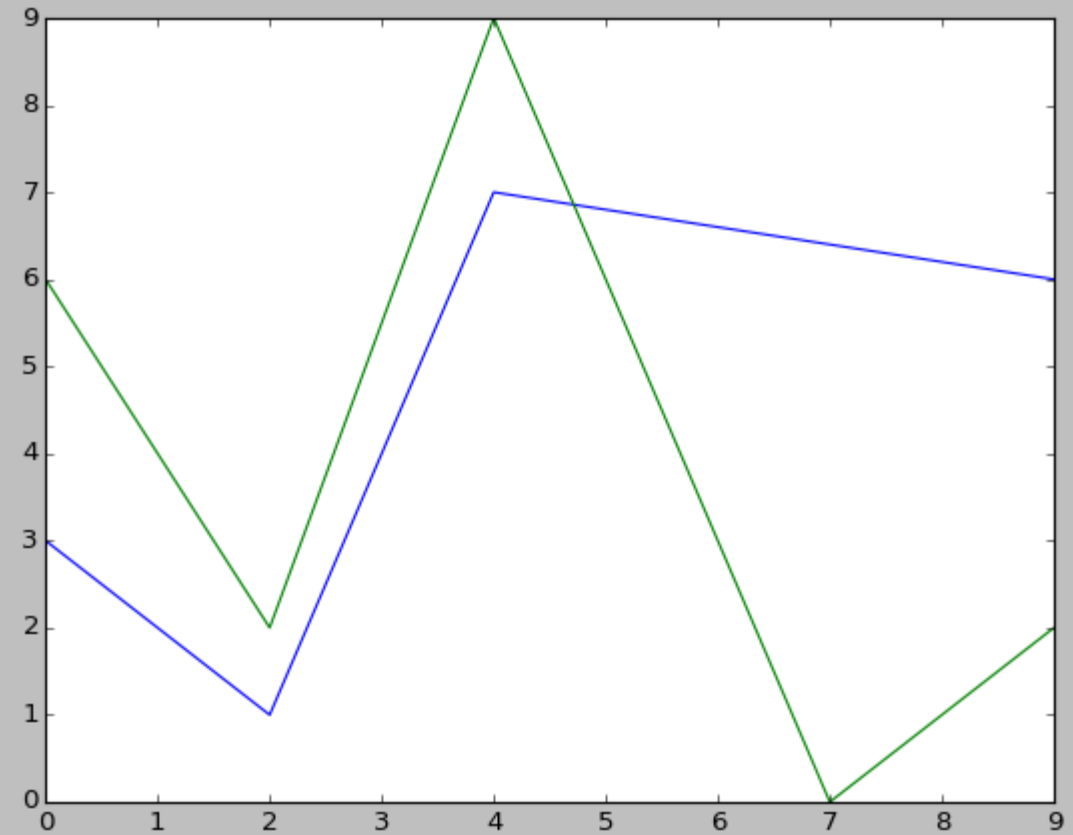




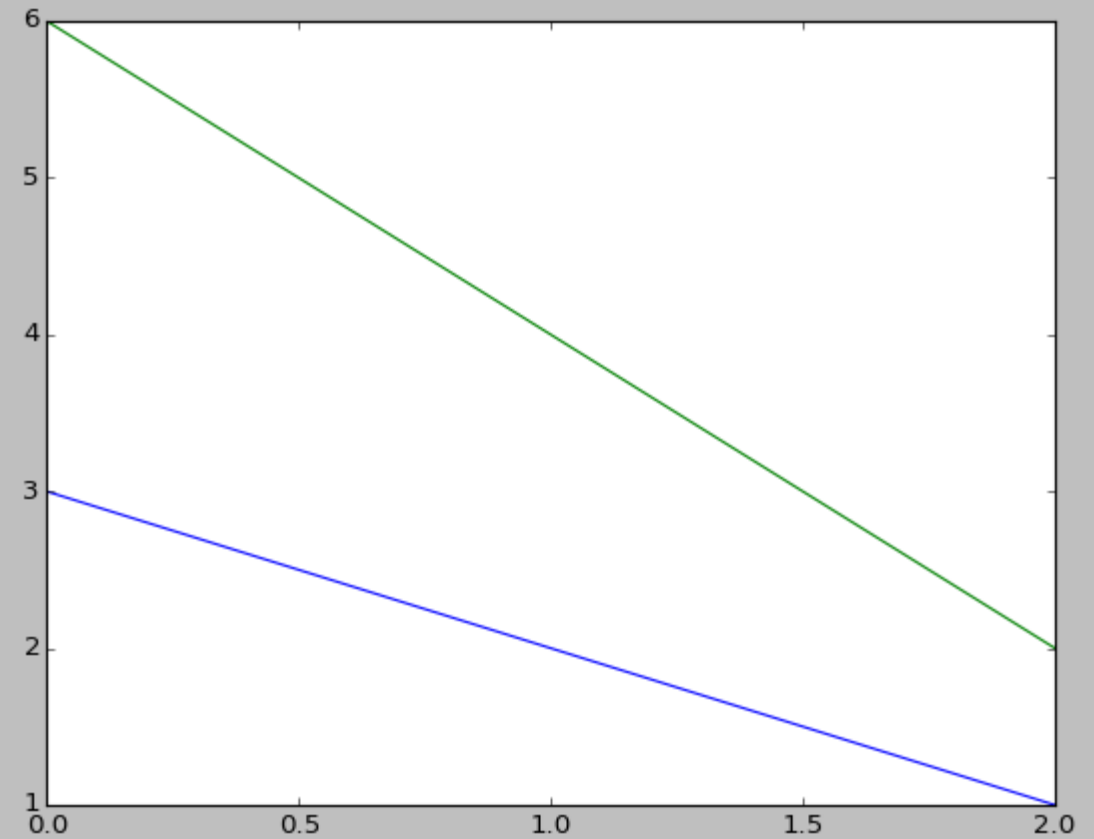
```
plt.plot([1, 2, 3, 4, 5], [3, 1, 4, 7])  
plt.show()
```

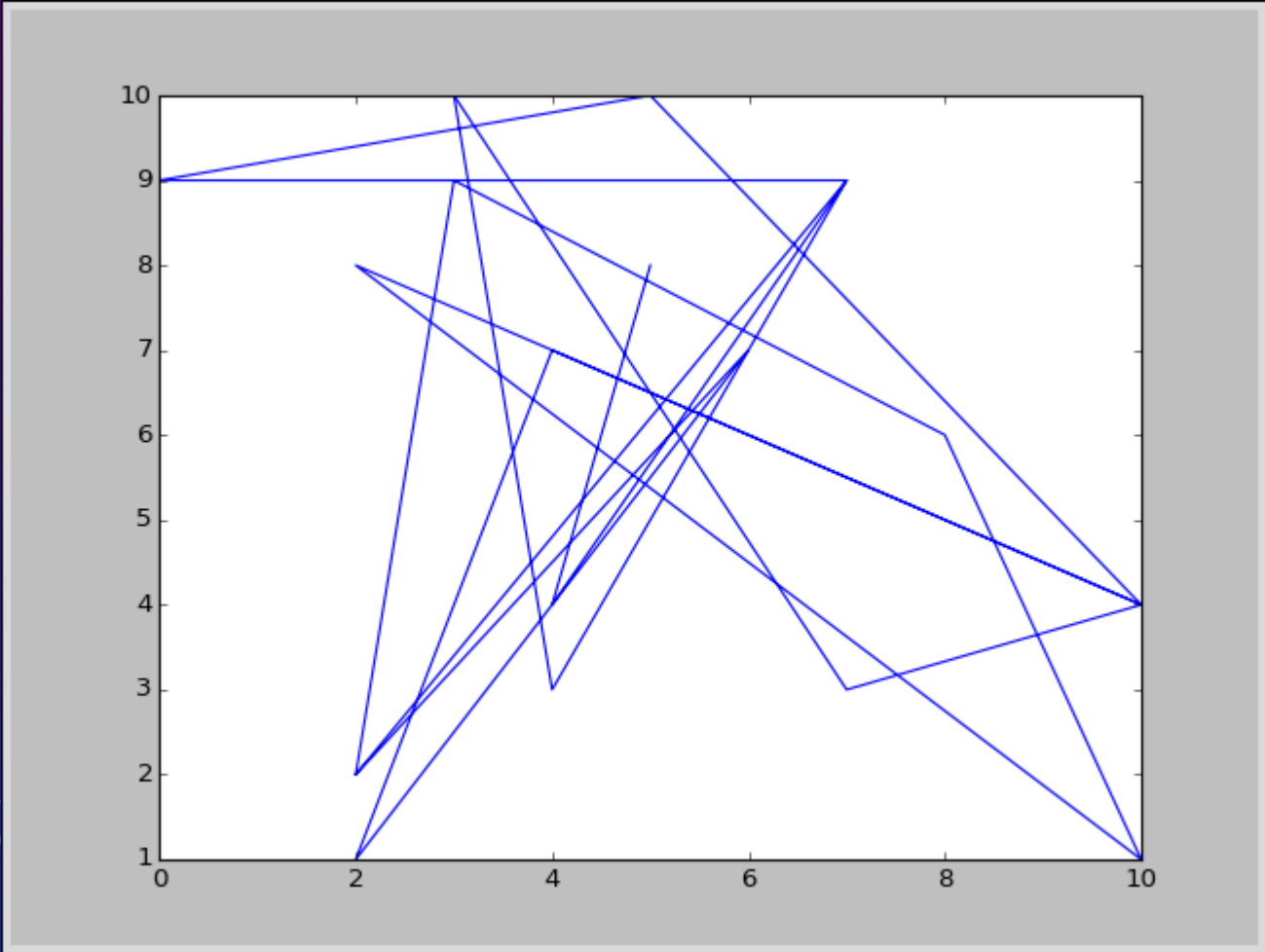
```
Traceback (most recent call last):  
  File "test.py", line 7, in <module>  
    plt.plot([1, 2, 3, 4, 5], [3, 1, 4, 7])  
  File "/usr/lib/python2.7/dist-packages/matplotlib/pyplot.py", line 3099, in plot  
    ret = ax.plot(*args, **kwargs)  
  File "/usr/lib/python2.7/dist-packages/matplotlib/axes/_axes.py", line 1374, in plot  
    for line in self._get_lines(*args, **kwargs):  
  File "/usr/lib/python2.7/dist-packages/matplotlib/axes/_base.py", line 303, in _grab_next_args  
    for seg in self._plot_args(remaining, kwargs):  
  File "/usr/lib/python2.7/dist-packages/matplotlib/axes/_base.py", line 281, in _plot_args  
    x, y = self._xy_from_xy(x, y)  
  File "/usr/lib/python2.7/dist-packages/matplotlib/axes/_base.py", line 223, in _xy_from_xy  
    raise ValueError("x and y must have same first dimension")  
ValueError: x and y must have same first dimension
```

```
plt.plot([0, 2, 3, 4, 9], [3, 1, 4, 7, 6])  
plt.plot([0, 2, 4, 7, 9], [6, 2, 9, 0, 2])  
  
plt.show()
```



```
plt.plot([0, 2], [3, 1], [0, 2], [6, 2])  
plt.show()
```

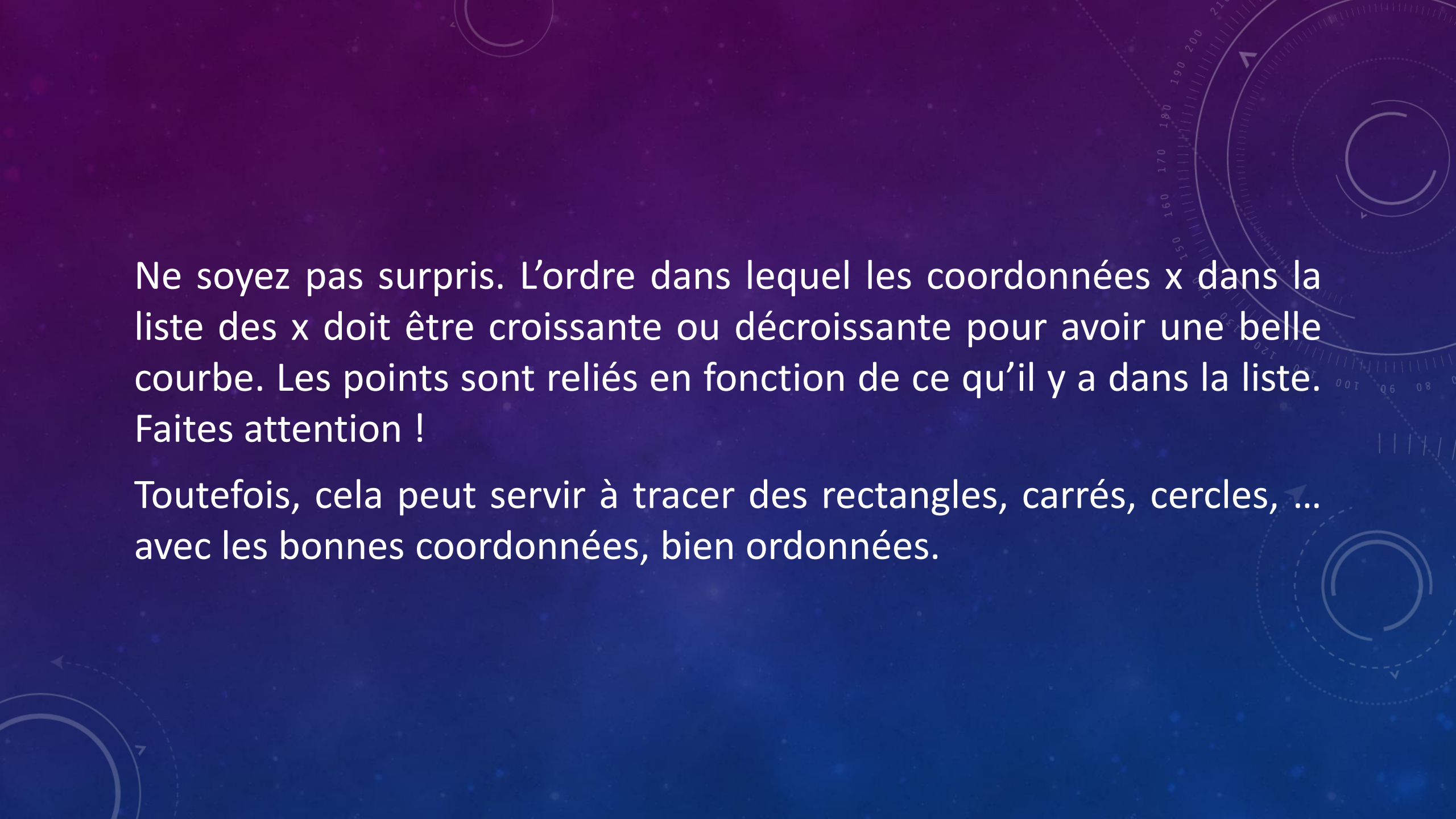




```
[5, 4, 7, 0, 5, 10, 4, 2, 6, 2, 3, 8, 10, 2, 10, 7, 3, 4, 7, 2]
[8, 4, 9, 9, 10, 4, 7, 1, 7, 2, 9, 6, 1, 8, 4, 3, 10, 3, 9, 2]
```







Ne soyez pas surpris. L'ordre dans lequel les coordonnées  $x$  dans la liste des  $x$  doit être croissante ou décroissante pour avoir une belle courbe. Les points sont reliés en fonction de ce qu'il y a dans la liste. Faites attention !

Toutefois, cela peut servir à tracer des rectangles, carrés, cercles, ... avec les bonnes coordonnées, bien ordonnées.

# LES ARGUMENTS

Savoir placer des points est certes remarquable, mais il est possible de modifier ce que vous avez placé sur votre graphe. Pour cela, il suffit de placer vos « modifications » à la suite des coordonnées de vos points, à la place des « ... ».

**`plt.plot(x, y, ..., ..., ...)`**

Voyons-les un par un.

# FORME DU POINT

Tout le monde sait ce qu'est un point. C'est une forme ronde, remplie. Comme une ligne, c'est un rectangle, de largeur très petite. Il est possible de modifier cette forme.

Il suffit de la spécifier la forme avec l'argument **marker** = « **forme** », ou simplement entre « ». Vous pouvez également utiliser les apostrophes ' '.

Attention ! Pas de majuscule à « marker ». Ceci est également valable pour tous les autres arguments.

# FORMES POSSIBLES

SYMBOLE				FORME			
Pour un point							
.				Un point			
o				Un point plus gros (un rond)			
+				Un plus			
*				Une étoile			
p	h	d	s	pentagone	hexagone	losange	carré
^	v	>	<	Un triangle (haut bas droite gauche)			
Pour une ligne (plusieurs points)							
-		solid		Une ligne pleine			
:		dotted		Des pointillées			
--		dashed		Des tirets			
-.		dashdot		Tiret puis point			



```
plt.plot(1, 2, "*")  
plt.show()
```



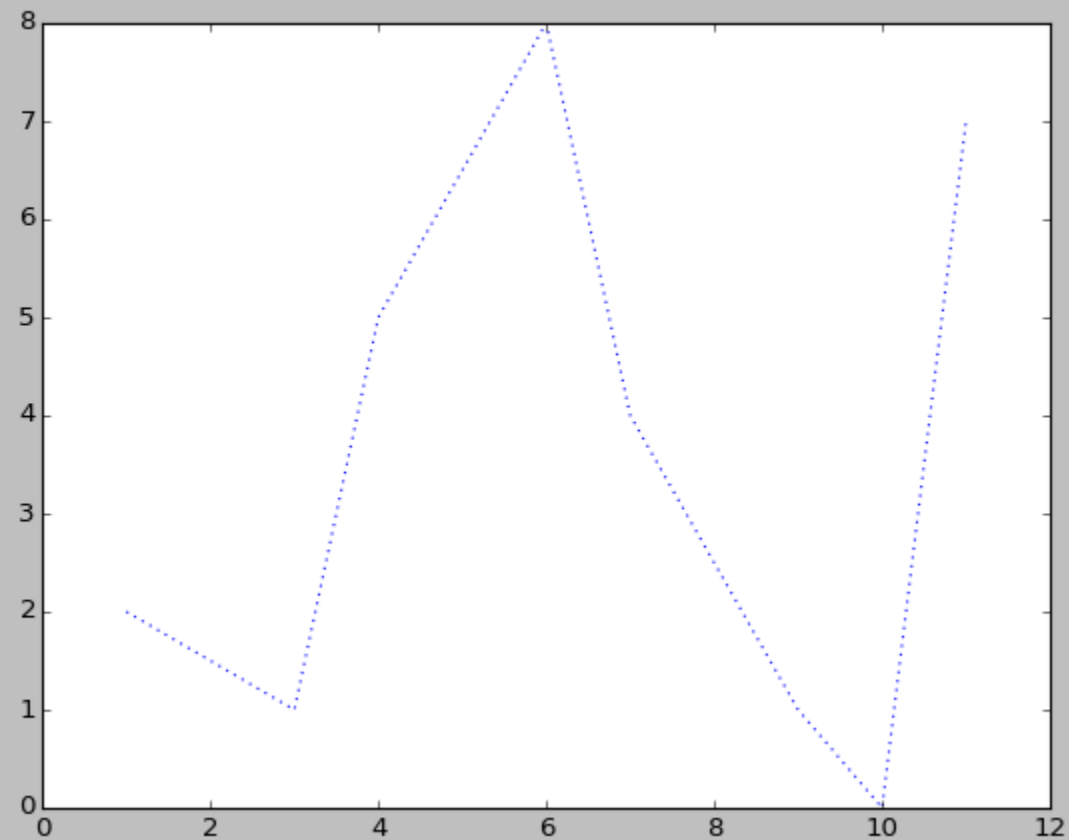
```
plt.plot(1, 2, "d")  
plt.show()
```



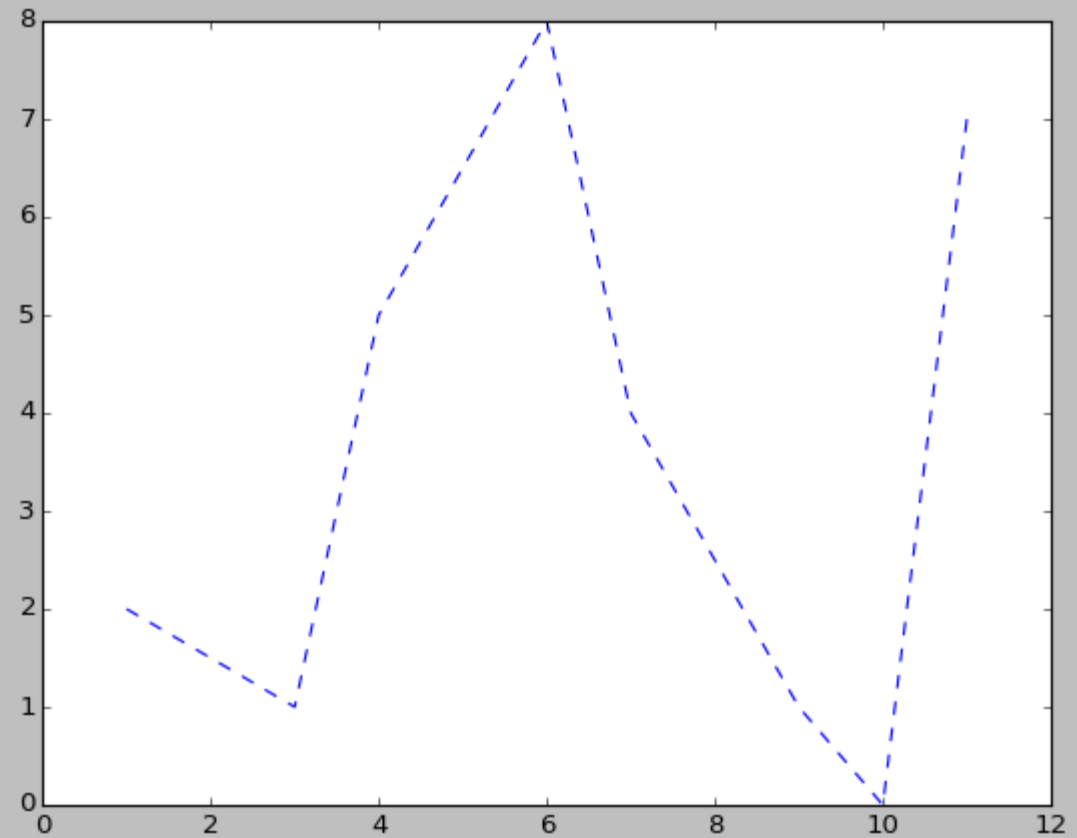
```
plt.plot(1, 2, "^")  
plt.show()
```



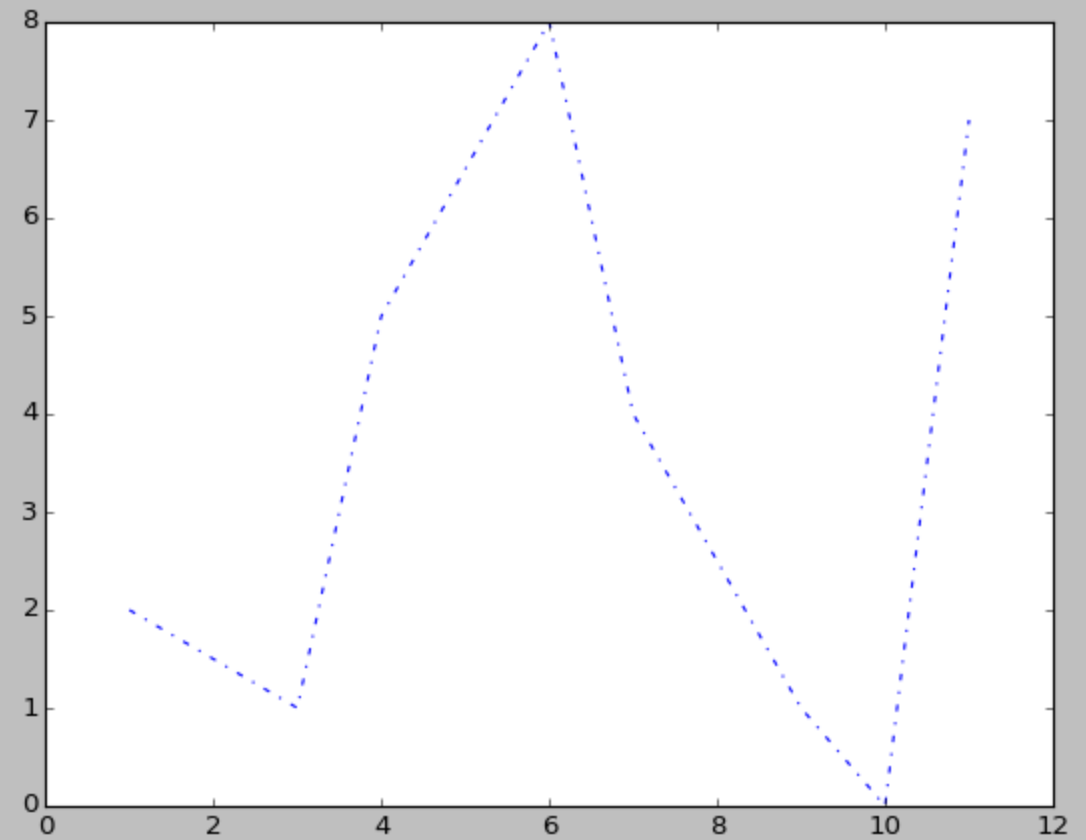
```
x = [1, 3, 4, 6, 7, 9, 10, 11]  
y = [2, 1, 5, 8, 4, 1, 0, 7]  
  
plt.plot(x, y, ls=":")  
plt.show()
```



```
x = [1, 3, 4, 6, 7, 9, 10, 11]  
y = [2, 1, 5, 8, 4, 1, 0, 7]  
  
plt.plot(x, y, "--")  
  
plt.show()
```



```
x = [1, 3, 4, 6, 7, 9, 10, 11]  
y = [2, 1, 5, 8, 4, 1, 0, 7]  
  
plt.plot(x, y, linestyle="-.")  
plt.show()
```



Vous savez maintenant comment tracer et changer le style d'une ligne ! Il suffit de spécifier avec **linestyle** = « » ou **ls** = « », le style de la ligne reliant les points, ou seulement entre « ».

```
plt.plot(x, y, "--")
```

Ou bien

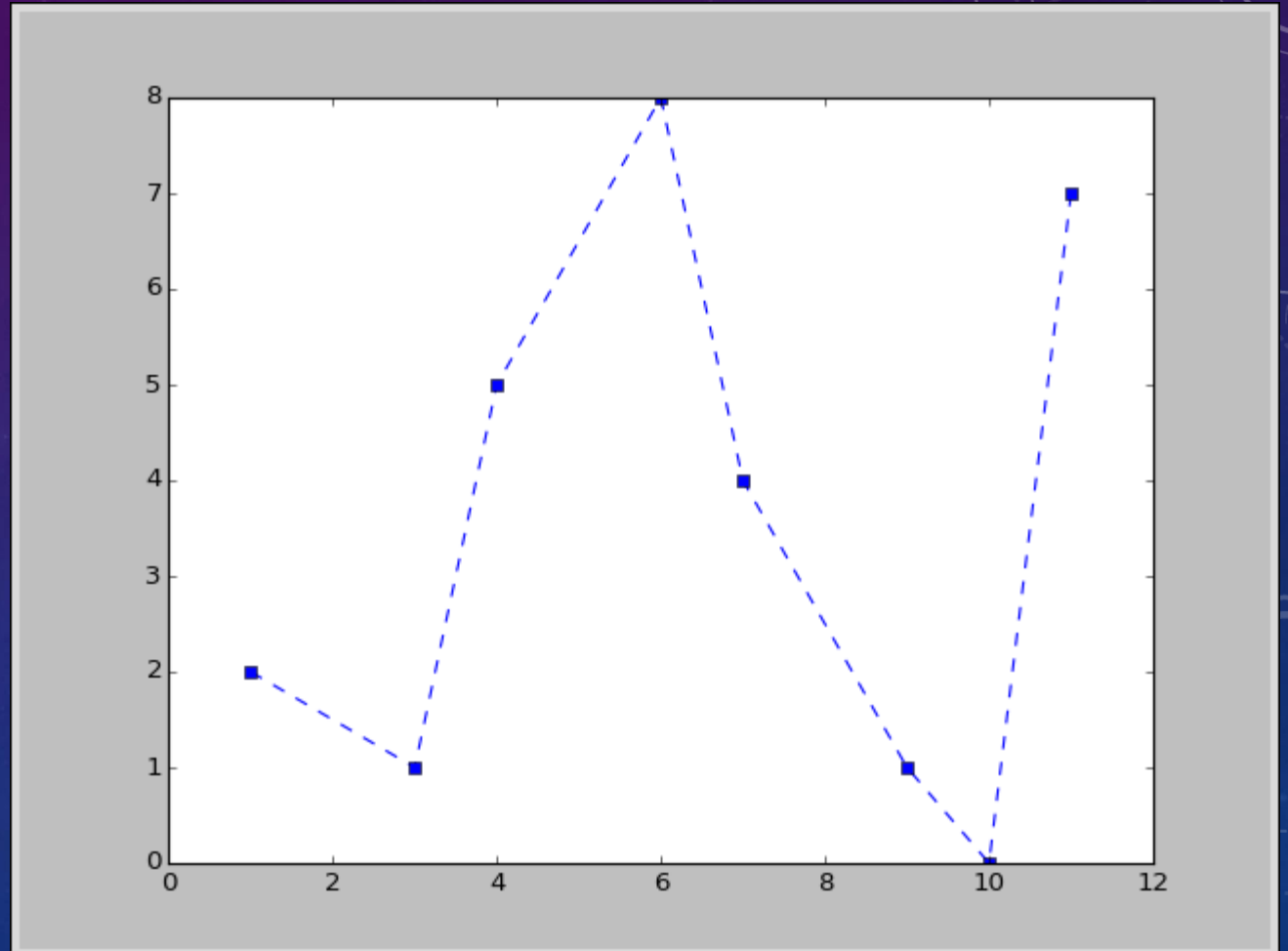
```
plt.plot(x, y, linestyle="-.")
```

Si vous souhaitez afficher chaque point en même temps que la ligne, faites figurer dans les arguments **marker** = « » ainsi que **linestyle** = « ».



# EXAMPLE

```
x = [1, 3, 4, 6, 7, 9, 10, 11]  
y = [2, 1, 5, 8, 4, 1, 0, 7]  
  
plt.plot(x, y, marker="s", linestyle="--")  
  
plt.show()
```



# COULEUR

Il est également possible de modifier la couleur de vos points et lignes. Il suffit de placer votre couleur entre les guillemets de l'argument **color** = « **string** » ou **c** = « **string** ».

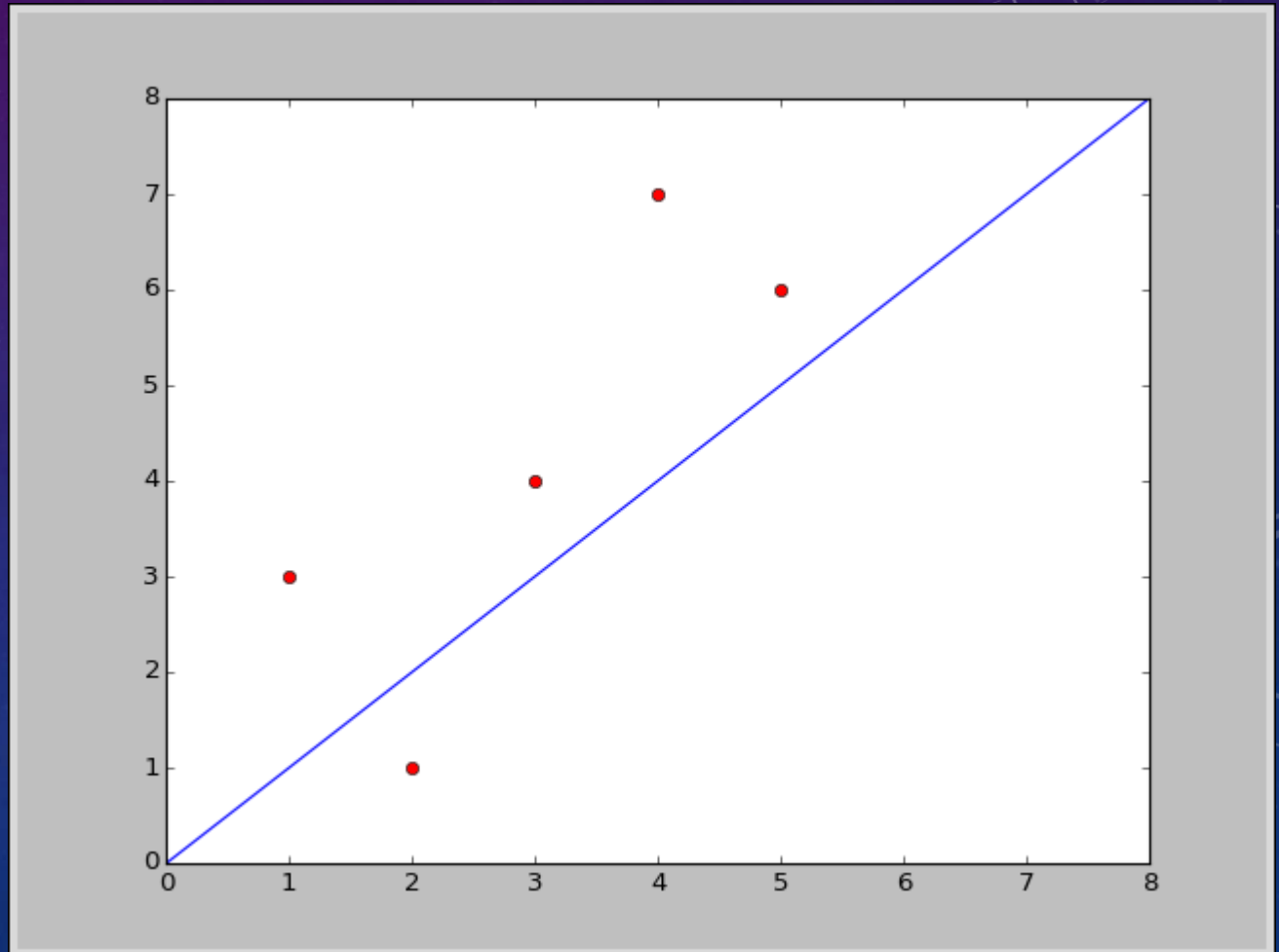
Note : vous pouvez insérer ou non des espaces de part et d'autre du signe « = », cela n'a pas de différence.

Note 2 : **c** = « » ne fonctionne pas tout le temps. Il faut écrire **color** = « ».

COULEUR	À TAPER ENTRE LES « » OU ‘’	
Bleu	b	blue
Vert	g	green
Rouge	r	red
Cyan	c	cyan
Magenta	m	magenta
Jaune	y	yellow
Noir	k	black
Blanc	w	white

# EXAMPLE

```
plt.plot([1, 2, 3, 4, 5], [3, 1, 4, 7, 6], "o", color="r")  
plt.plot([0, 8], [0, 8], c="b")  
  
plt.show()
```





# ENCORE PLUS DE COULEURS

Si le choix de couleurs deux diapos avant vous est insuffisant, il est possible d'en avoir plus.

Vous pouvez passer à **color** = « » un tuple (ou une liste) de 3 valeurs entières ou réelles comprises entre 0 et 1. Vous l'aurez compris, un pour le niveau de rouge, un pour le niveau de vert, et le dernier pour le bleu. À vous d'entrer les bonnes valeurs pour obtenir la bonne couleur.

Normalement vous devez connaître ce format pour les couleurs mais avec des valeurs allant de 0 à 255. C'est la même chose sauf qu'il faut voir cela en pourcentage. 0 correspond à 0, 1 correspond à 255. À vous de faire la conversion pour avoir la bonne couleur (valeur sur 255 divisée par 255 pour l'avoir entre 0 et 1).

Vous pouvez également entrer un code HTML dont le format est : #XXXXXX, avec pour chaque X un chiffre ou une lettre majuscule (c'est une chaîne de caractères (donc entre « » ou '')).

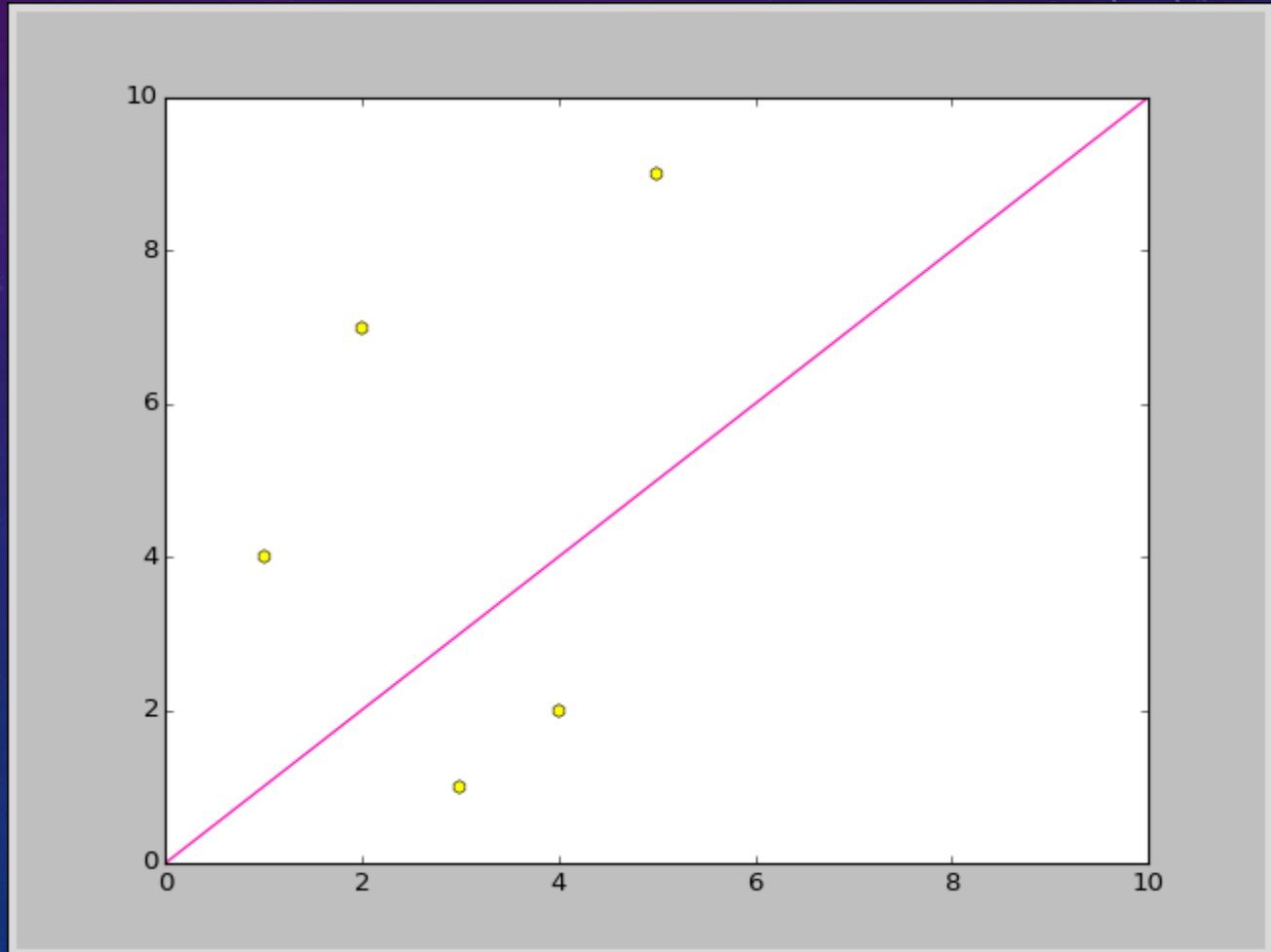
Quelques exemples seront donnés dans la diapos suivante, mais voici un site web où vous pourrez trouver votre bonheur : <http://html-color-codes.info/>

Couleur	RGB 0-255	RGB 0-1	Code HTML
Bleu	(0 ,0, 255)	(0, 0, 1)	#0000FF
Rouge	(255, 0, 0)	(1, 0, 0)	#FF0000
Vert	(0, 255, 0)	(0, 1, 0)	#00FF00
Noir	(0, 0, 0)	(0, 0, 0)	#000000
Blanc	(255, 255, 255)	(1, 1, 1)	#FFFFFF
Jaune	(255, 255, 0)	(1, 1, 0)	#FFFF00
Bleu turquoise	(0, 255, 255)	(0, 1, 1)	#00FFFF
Rose	(255, 0, 175)	(1, 0, 0.68)	#FF00AF
...	...	...	...

<http://html-color-codes.info/>

# ILLUSTRATION

```
plt.plot([0, 10], [0, 10], color='#FF00AF')  
plt.plot([1, 2, 3, 4, 5], [4, 7, 1, 2, 9], "h", color=(1, 1, 0))  
  
plt.show()
```



## ASTUCE

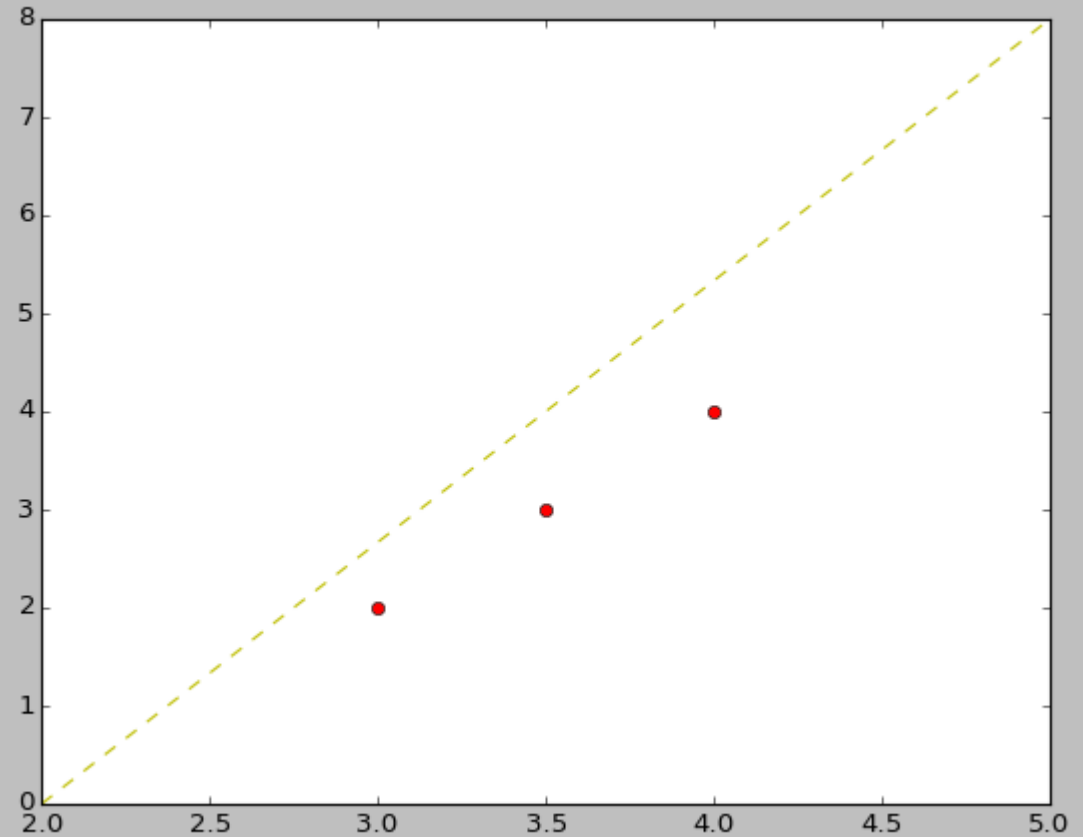
Si vous souhaitez tracer une ligne pleine rouge par exemple, au lieu d'écrire **plt.plot(x, y, marker = « - », c = « r »)**, vous pouvez utiliser la forme suivante :

**plt.plot(x, y, « r- »)**

Avouez que c'est plus simple ? Cela est bien sur possible avec toutes les couleurs et formes.

# EXAMPLE

```
plt.plot([2, 5], [0, 8], "y--")  
plt.plot([3, 3.5, 4], [2, 3, 4], "ro")  
plt.show()
```





## LARGEUR DE LA LIGNE

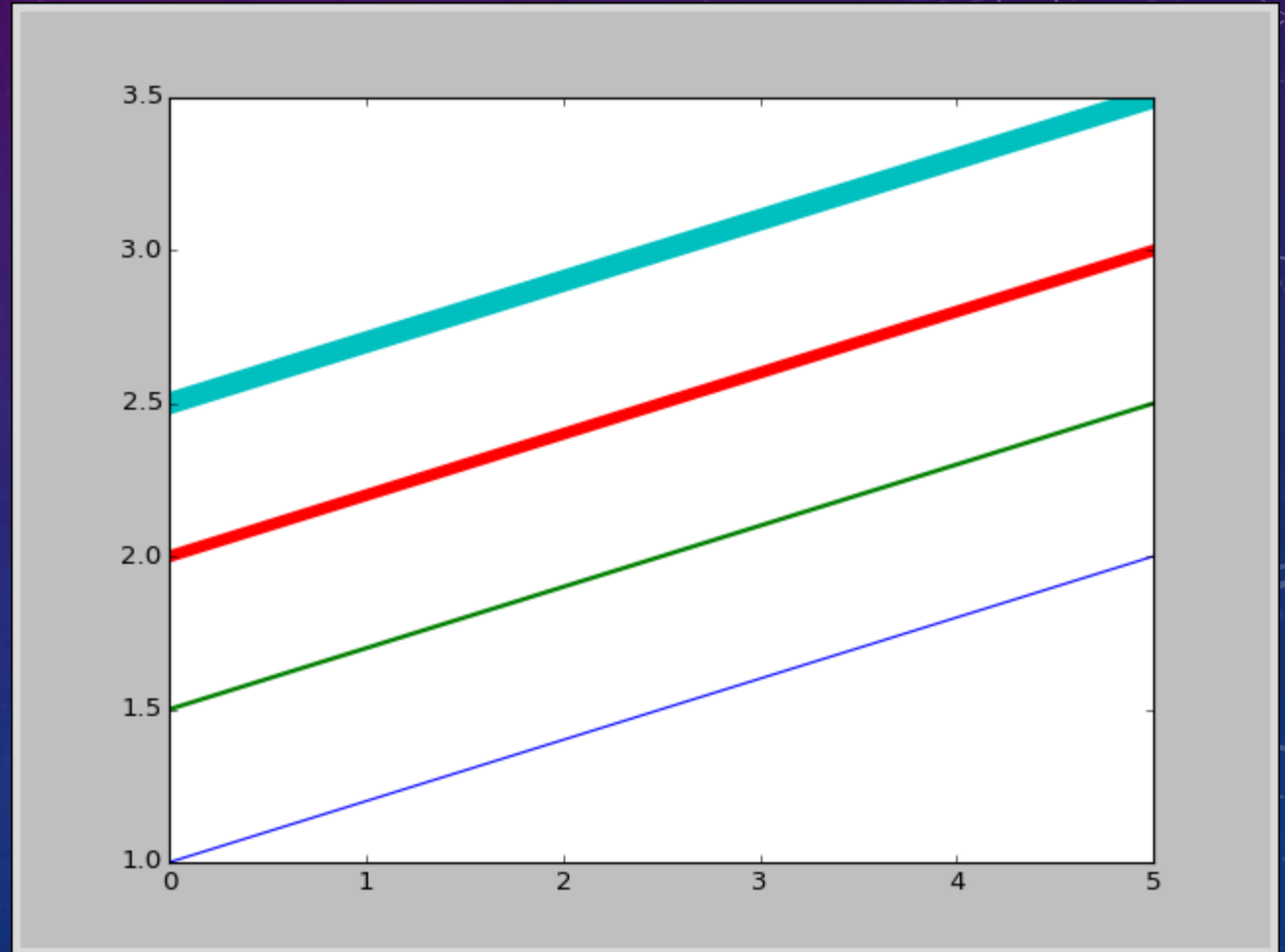
**linewidth = int/float** ou bien **lw = int/float**

Par défaut, l'épaisseur de la ligne est de 1.

# EXAMPLE

```
plt.plot([0, 5], [1, 2], linewidth=1)
plt.plot([0, 5], [1.5, 2.5], lw=2)
plt.plot([0, 5], [2, 3], lw=5.5)
plt.plot([0, 5], [2.5, 3.5], linewidth=10)

plt.show()
```



# Compléments additionnels

The background is a dark blue gradient with faint, light blue technical diagrams. On the right side, there is a large circular scale with degree markings from 0 to 210. Below it, there are concentric circles with arrows indicating a clockwise direction. In the bottom left corner, there is another circular diagram with a dashed arrow pointing counter-clockwise.

- Ajouter du texte
- Compléments texte
- Modification des axes
  - limites
  - graduation
- Enregistrer l'image
- Autres

# Ajouter du texte

Note : ne pas mettre d'accent sur les lettres



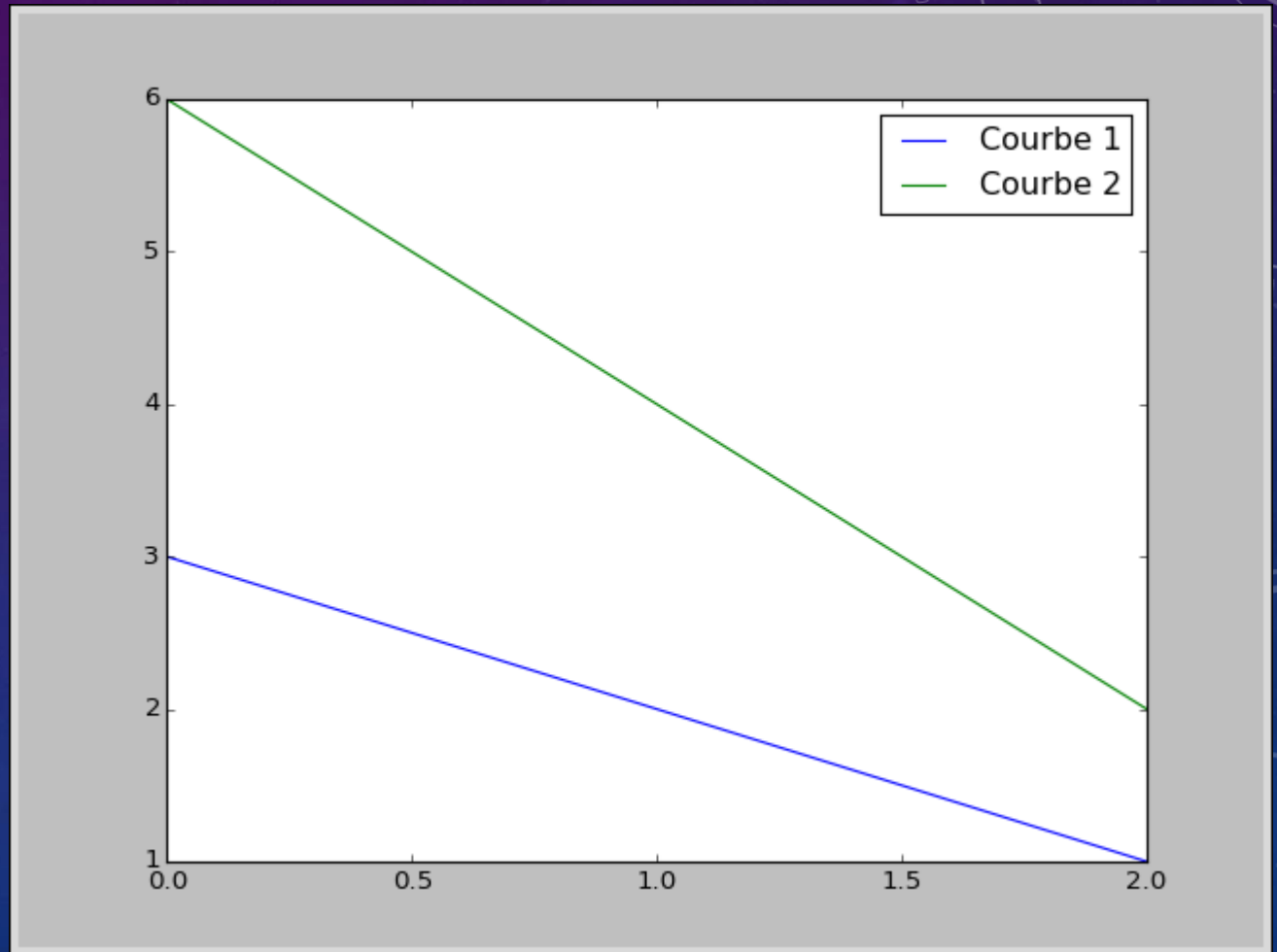
# LÉGENDE D'UNE SÉRIE DE DONNÉES

**label = « string »**

Pour afficher la légende, ajouter la commande **plt.legend()**

# EXAMPLE

```
plt.plot([0, 2], [3, 1], label="Courbe 1")  
plt.plot([0, 2], [6, 2], label="Courbe 2")  
plt.legend()  
  
plt.show()
```



# Quelques arguments de plt.legend()

- **shadow = True/False**

Si True, affiche l'ombre du cadre contenant les légendes.

- **title = « »**

Titre des légendes.

- **fontsize = int/float**

Taille du cadre des légendes int/float ou xx-small, x-small, small, medium, large, x-large, xx-large.

- **loc = « » ou « »**

Localisation du cadre des légendes.

# LOCALISATION

Localisation	À taper entre les « » ou ‘’	
Meilleur endroit selon l'ordinateur	best	0
En haut à droite	upper right	1
En haut à gauche	upper left	2
En bas à gauche	lower left	3
En bas à droite	lower right	4
À droite (au centre à droite)	right	5
Au centre à gauche	centre left	6
Au centre à droite	center right	7
En bas au centre	lower center	8
En haut au centre	upper center	9
Au centre	center	10

# TITRE PRINCIPAL ET TITRE DE L'AXE DES X ET DES Y

- Titre principal

**plt.title(« un titre »)**

Par défaut, le titre se place tout en haut au milieu du graphe. Vous pouvez l'écrire sur plusieurs lignes, en ajoutant des saut de ligne « \n » si votre titre est long.

- Axe des x

**plt.xlabel(« titre de l'axe des x »)**

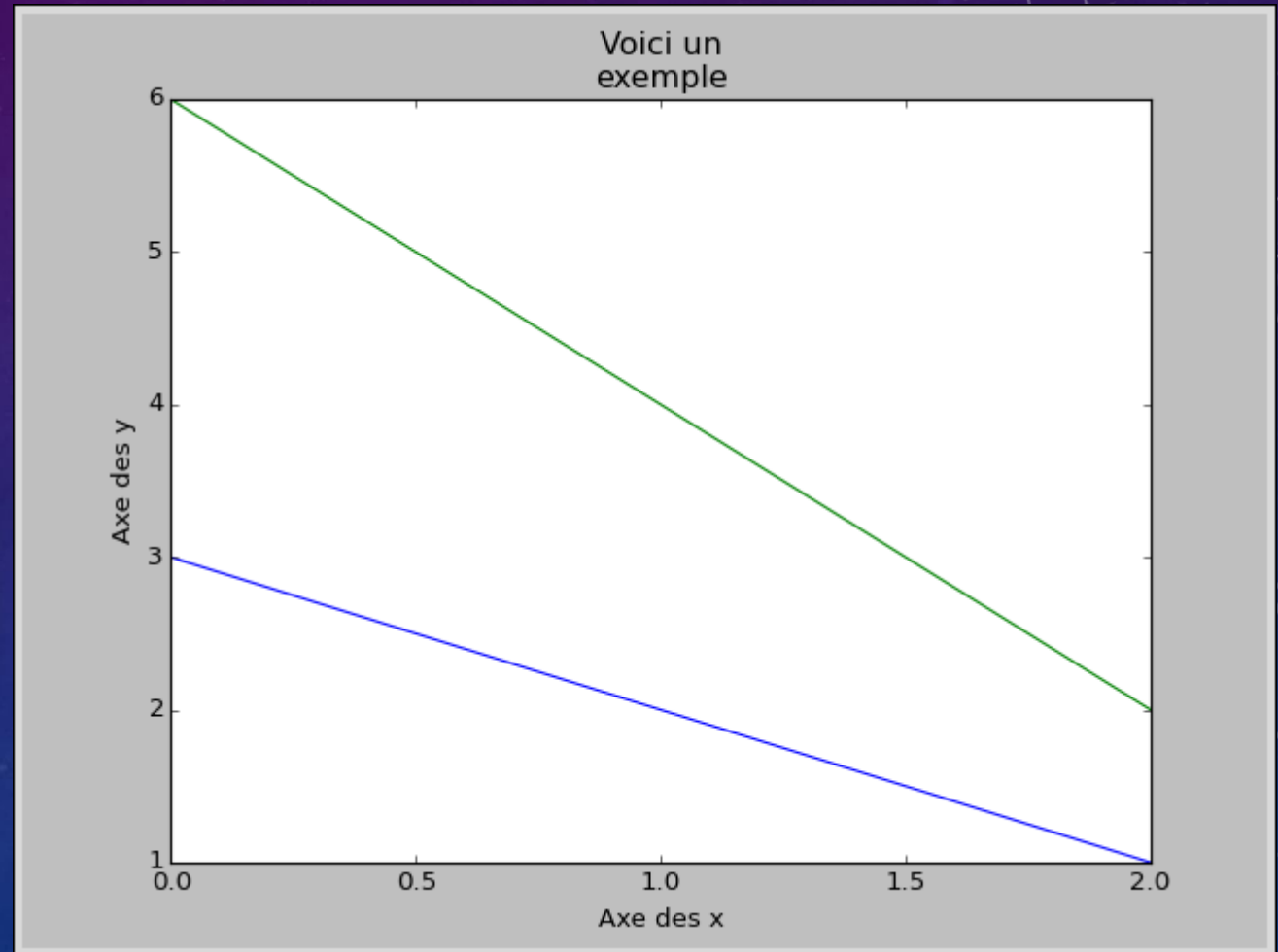
- Axe des y

**plt.ylabel(« titre de l'axe des y »)**



# EXEMPLE

```
plt.plot([0, 2], [3, 1])  
plt.plot([0, 2], [6, 2])  
  
plt.title("Voici un\nexemple")  
plt.xlabel("Axe des x")  
plt.ylabel("Axe des y")
```



# TEXTE BRUT

**`plt.text(x, y, « votre texte »)`**

La première lettre de votre texte se positionnera aux coordonnées (x, y). Vous pouvez modifier cela en ajoutant un argument qui vous sera expliqué plus loin.

Pour afficher les coordonnées de chaque point sur le graphe, la hauteur des barres de vos histogrammes, vous devez utiliser cette commande.

# Placer du texte à proximité de vos points ou au dessus de vos barres

- Pour les points

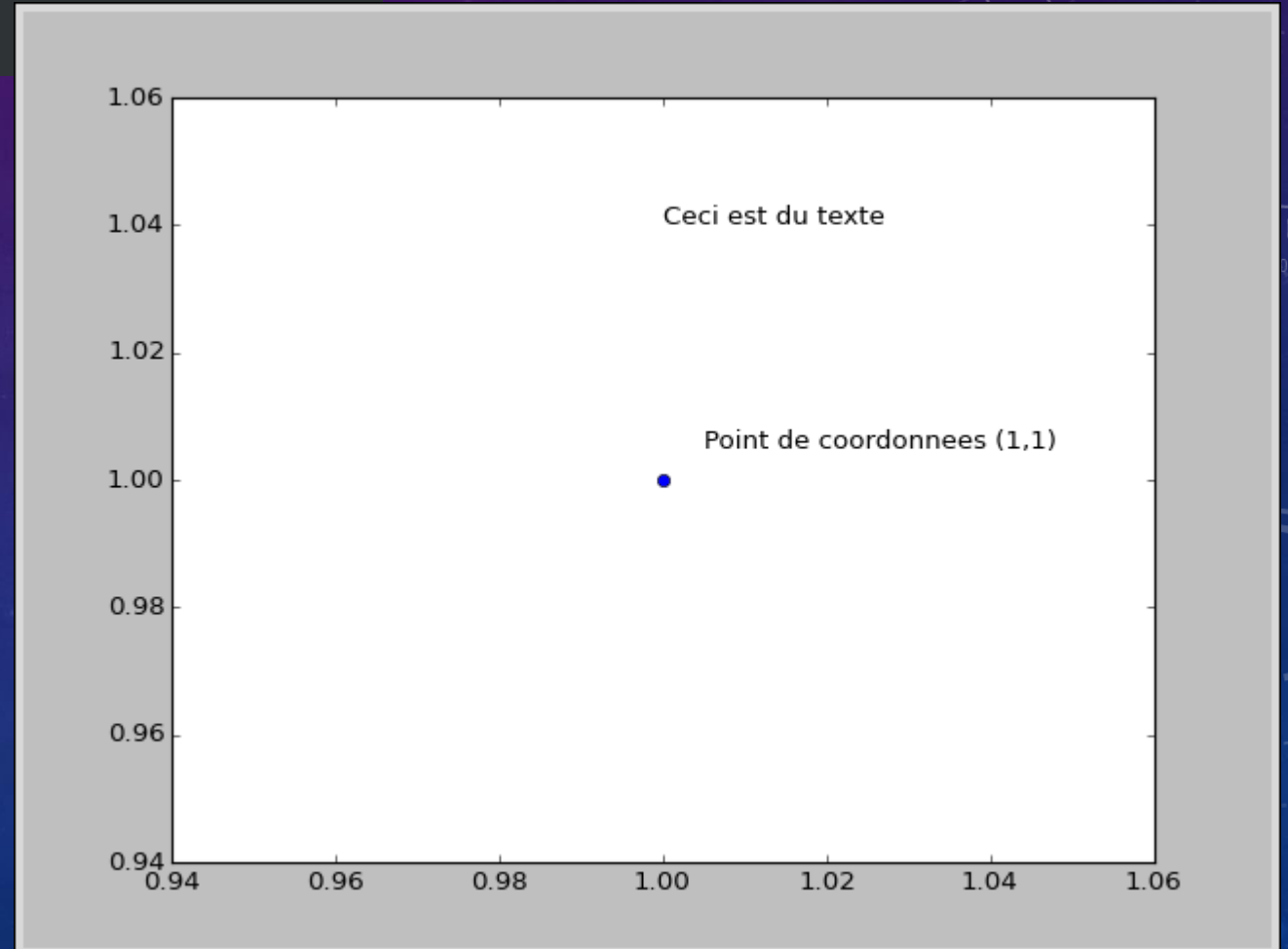
Il suffit de reprendre les coordonnées  $x$  et  $y$  de chaque point et de les modifier légèrement de telle sorte que le texte soit placé à gauche ou à droite en modifiant la valeur de  $x$ , en haut ou en bas (valeur de  $y$ ), ou diagonalement ( $x$  et  $y$ ). À vous ensuite de juger en affichant le graphe si vos textes sont bien placés. Le cas échéant, changez les coordonnées du texte, puis relancer.

- Pour les barres

Même principe. Pour placer la valeur de la hauteur d'une barre au dessus de celle-ci ou à côté, jouez sur la valeur de  $y$  (dans le cas de barres verticales) ou  $x$ , si barres horizontales. Puis l'autre coordonnée pour associer la valeur à la bonne barre. Nous verrons cela plus en détails dans la partie histogramme/diagrammes en barres.

# EXEMPLE

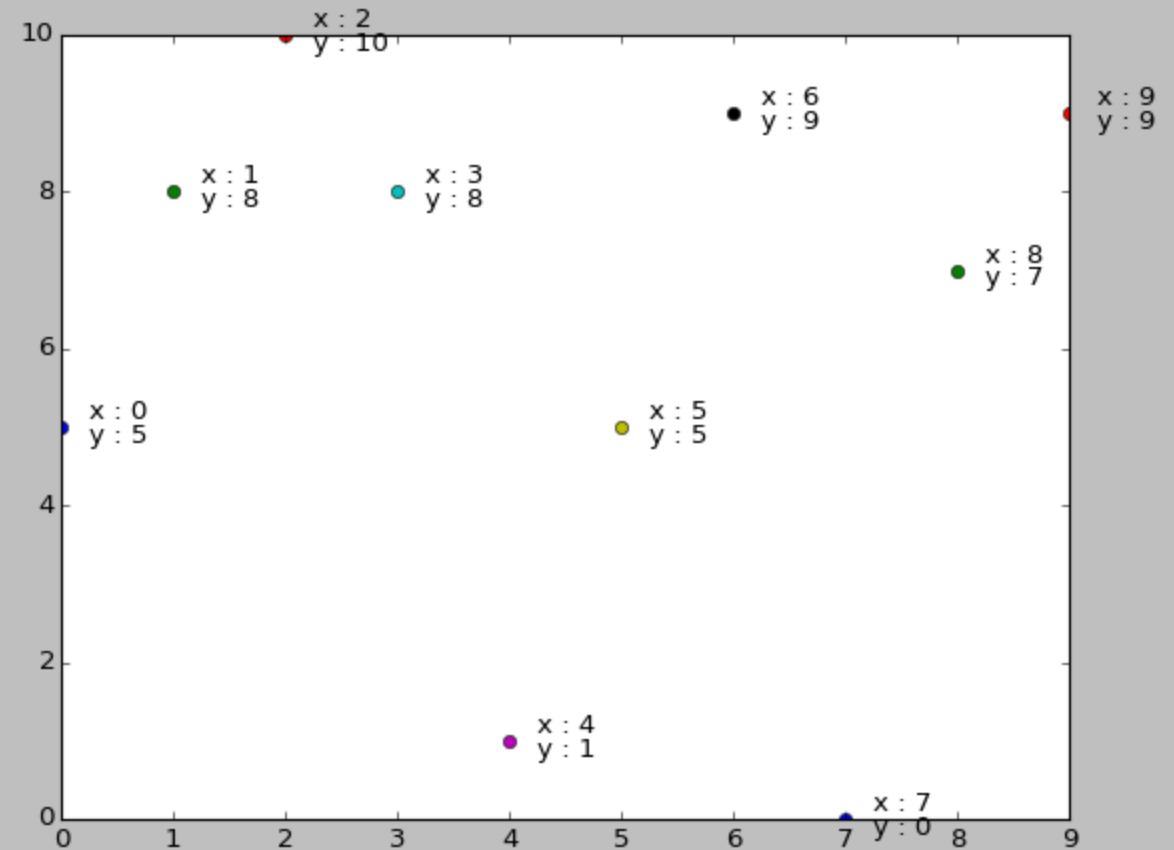
```
x, y = 1, 1  
plt.plot(x, y, "o")  
  
plt.text(1.005, 1.005, "Point de coordonnees (" + str(x) + "," + str(y) + ")")  
plt.text(1, 1.04, "Ceci est du texte")  
  
plt.show()
```



```
x = range(10)
y = [random.randint(0,10) for i in range(10)]

for a,b in zip(x,y) :
    plt.plot(a, b, "o")
    plt.text(a+0.25, b+0.1, "x : "+str(a))
    plt.text(a+0.25, b-0.2, "y : "+str(b))

plt.show()
```





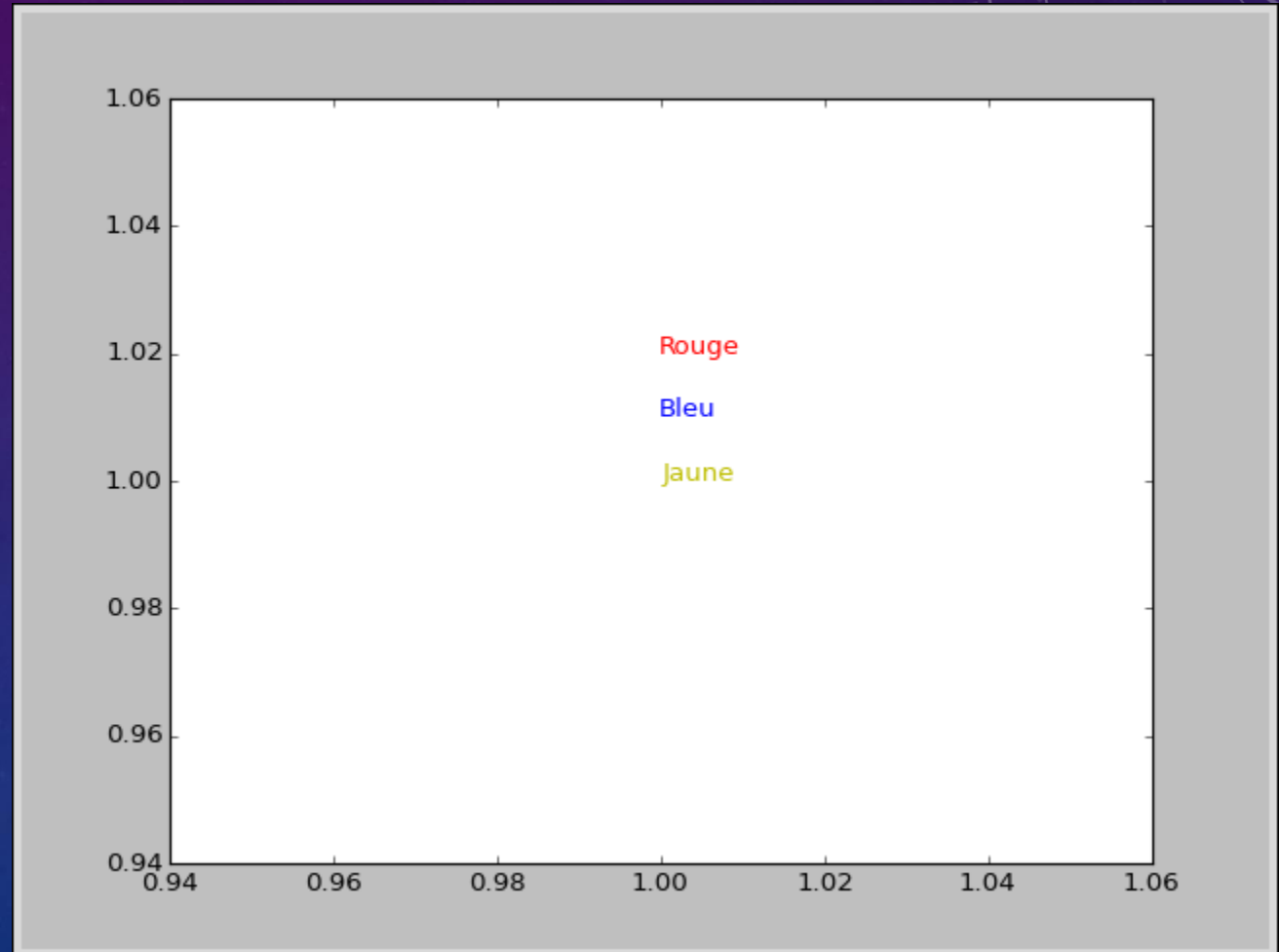
# Compléments texte

# COULEUR

Vous pouvez modifier la couleur de votre texte de la même façon que vous modifier la couleur de vos points et lignes avec **color = « »**.

# EXEMPLE

```
plt.plot(1, 1)  
plt.text(1, 1, "Jaune", color="y")  
plt.text(1, 1.01, "Bleu", color="b")  
plt.text(1, 1.02, "Rouge", color="r")  
plt.show()
```



# TAILLE DU TEXTE

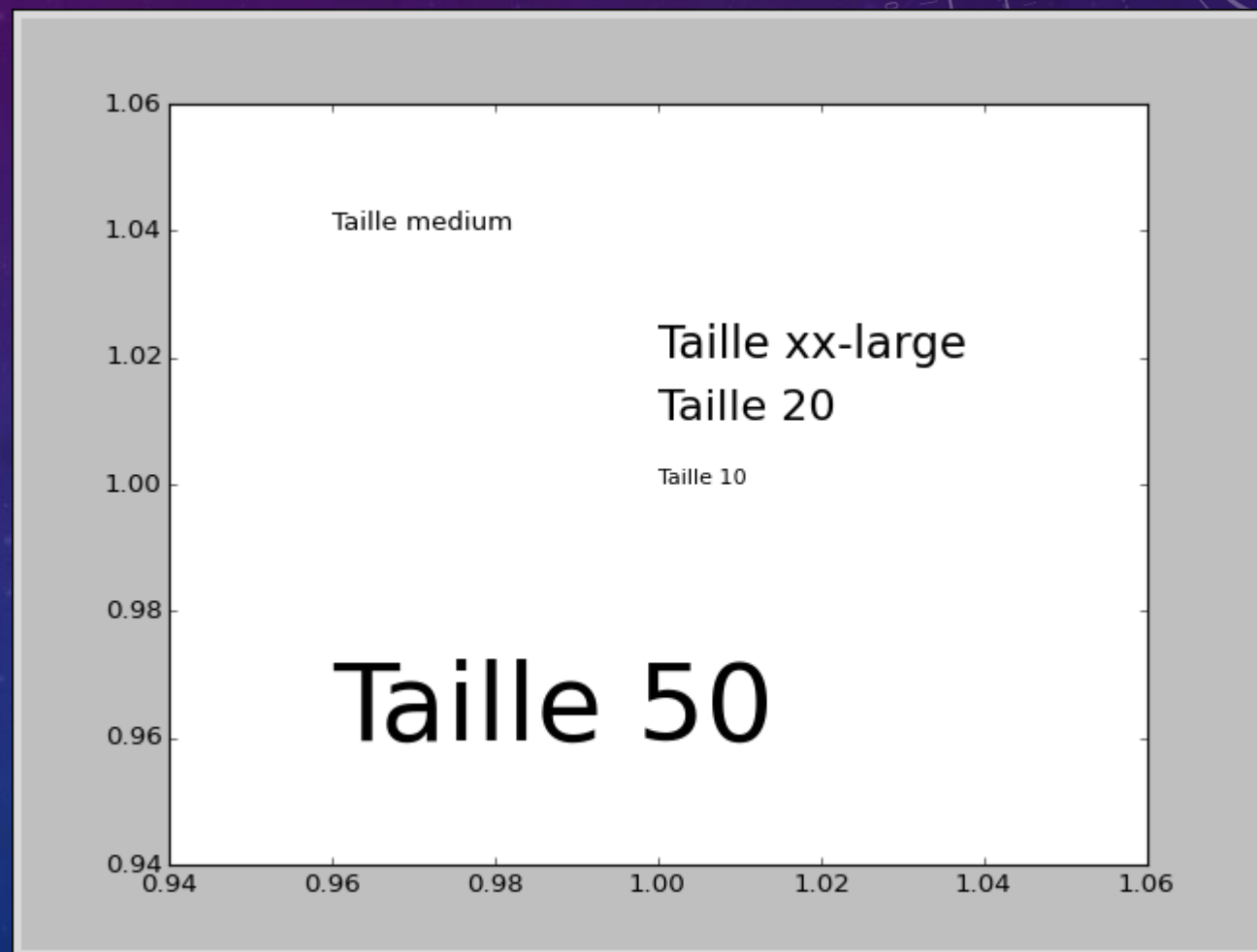
Pour modifier la taille du texte, il faut utiliser **size = int/float**, avec int/float, la taille du texte. Vous pouvez également utiliser xx-small, x-small, small, medium, large, x-large, xx-large (avec des « » ou des ' ').

# EXEMPLE

```
plt.plot(1, 1)

plt.text(1, 1, "Taille 10", size=10)
plt.text(1, 1.01, "Taille 20", size=20)
plt.text(1, 1.02, "Taille xx-large", size="xx-large")
plt.text(0.96, 0.96, "Taille 50", size=50)
plt.text(0.96, 1.04, "Taille medium", size='medium')

plt.show()
```





# ORIENTATION DU TEXTE

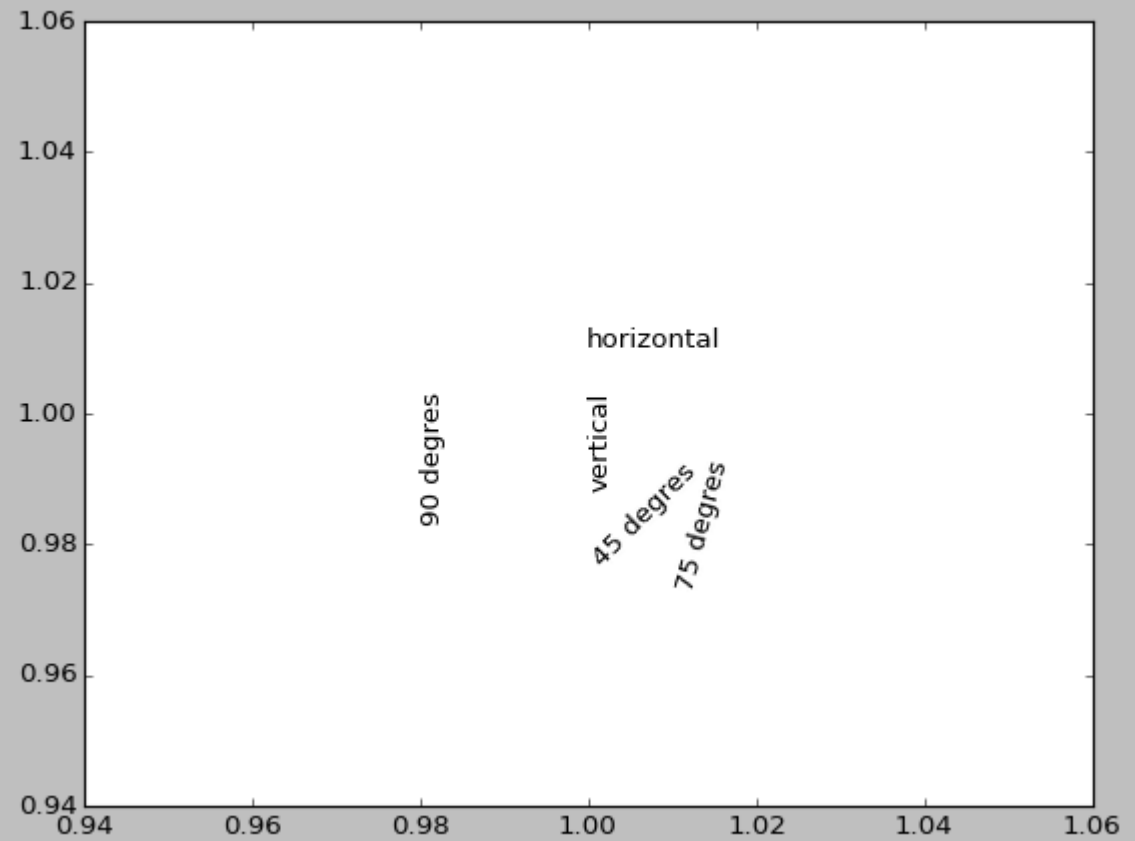
Vous pouvez écrire votre texte horizontalement, verticalement, ou selon un certain angle (en degrés) que vous précisez avec **rotation = int/float** avec int/float votre angle. Vous pouvez également écrire « horizontal » ou « vertical ». Par défaut le texte est écrit horizontalement (angle de 0 degré).

# EXAMPLE

```
plt.plot(1, 1)

plt.text(1, 1.01, "horizontal")
plt.text(1, 1, "vertical", rotation="vertical")
plt.text(0.98, 1, "90 degrees", rotation=90)
plt.text(1, 0.99, "45 degrees", rotation=45)
plt.text(1.01, 0.99, "75 degrees", rotation=75)

plt.show()
```



# STYLE

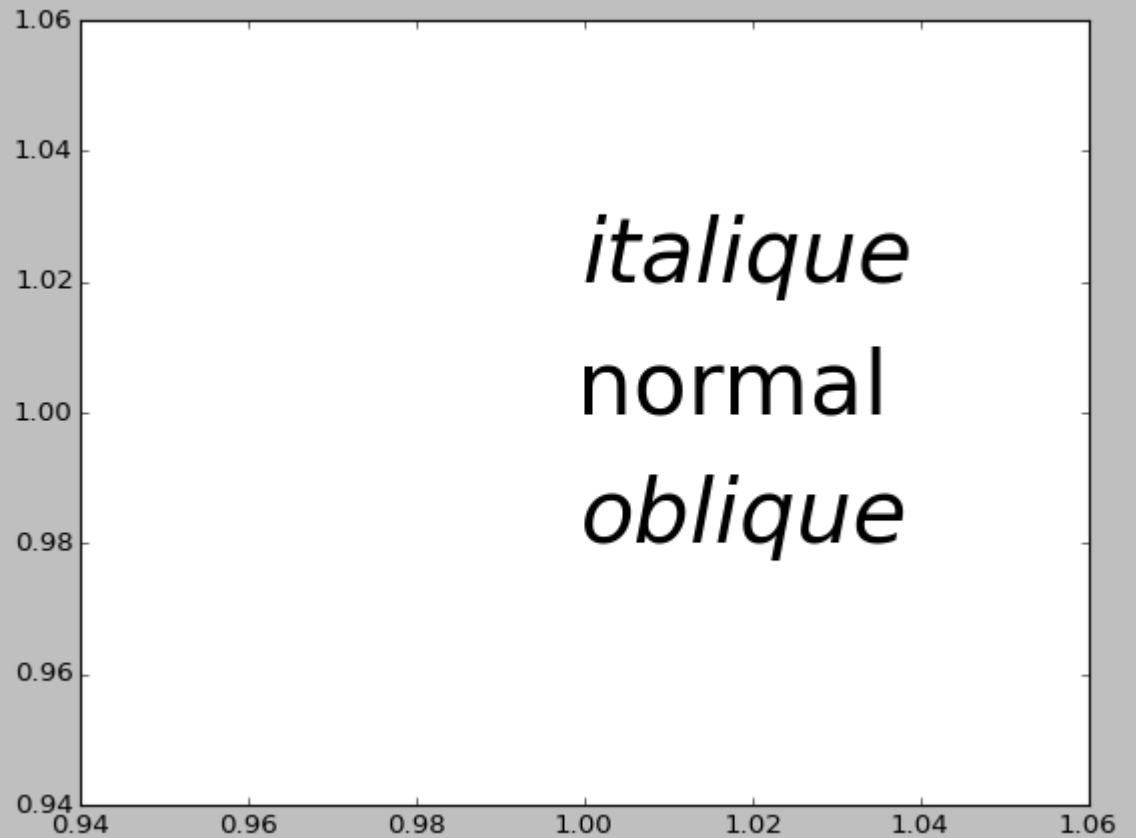
Par défaut votre texte est écrit normalement (normal, comme ici).  
Vous pouvez écrire en italique (italic) ou en oblique (oblique) avec  
**style = « »**.

# EXAMPLE

```
plt.plot(1, 1)

plt.text(1, 1, "normal", size=40)
plt.text(1, 1.02, "italique", style='italic', size=40)
plt.text(1, 0.98, "oblique", style="oblique", size=40)

plt.show()
```



# ALIGNEMENT PAR RAPPORT AUX COORDONNÉES (X, Y)

La première lettre de votre texte se situe au point de coordonnées (x, y). Vous pouvez modifier la position du texte par rapport à ces coordonnées en précisant l'alignement horizontal et vertical du texte.

- Alignement vertical

**verticalalignment** = « » ou bien **va** = « »

Choix possibles : centre (center), haut (top, le point est en haut), bas (bottom, point en bas)

- Alignement horizontal

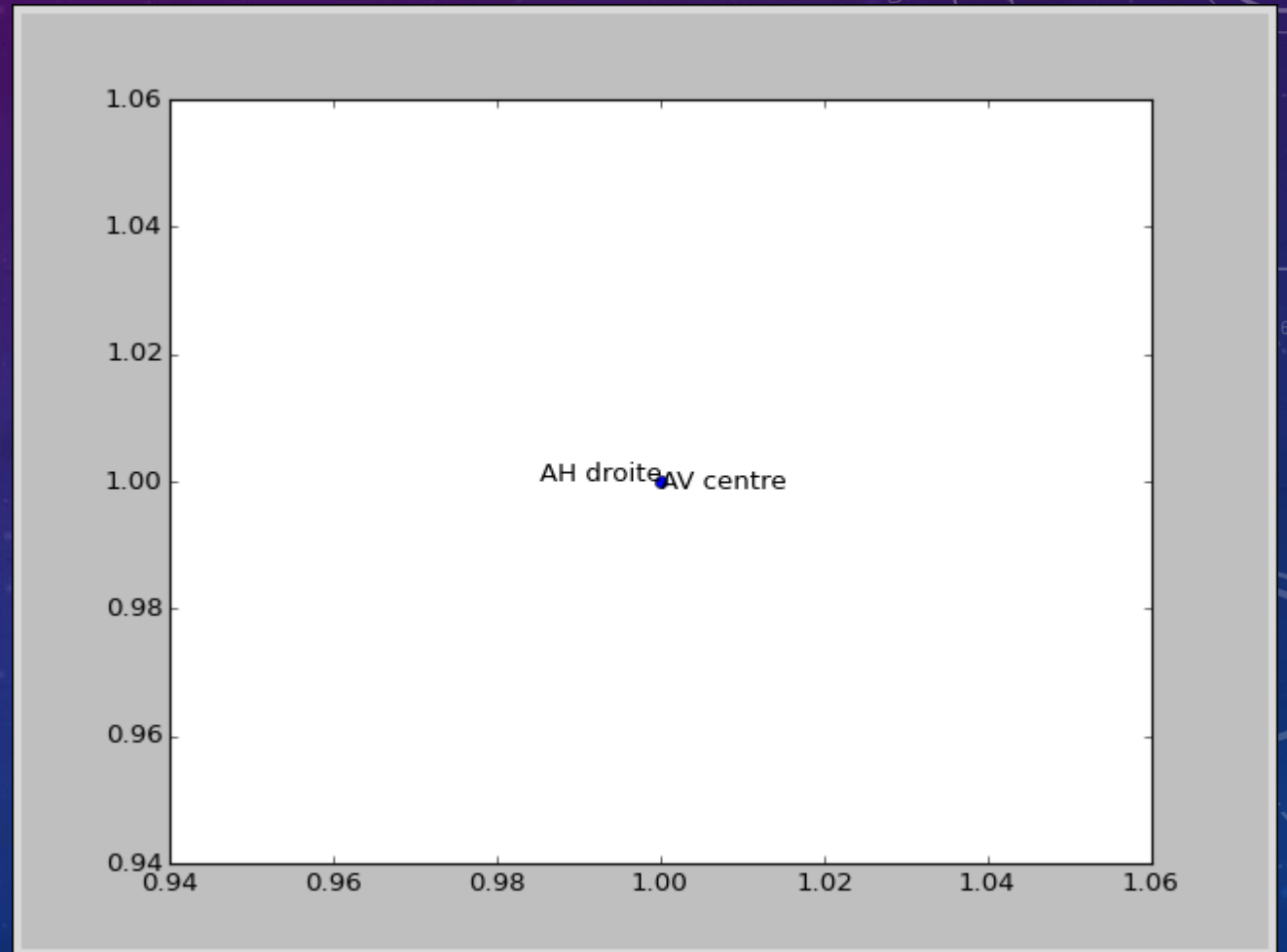
**horizontalalignment** = « » ou bien **ha** = « »

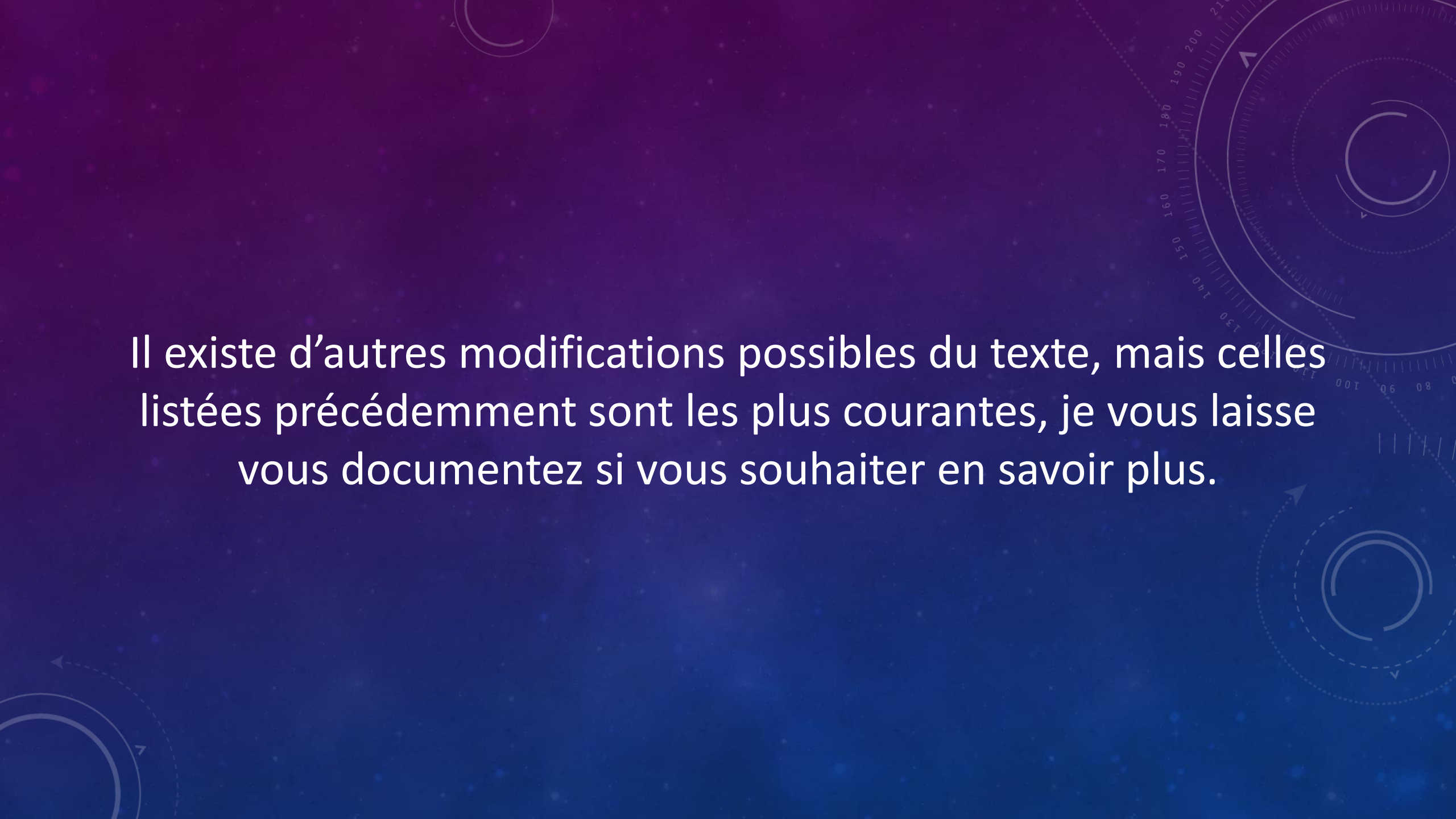
Choix possibles : centre (center), droite (right, point à droite), gauche (left, point à gauche)



# EXAMPLE

```
plt.plot(1, 1, "o")  
plt.text(1, 1, "AV centre", va='center')  
plt.text(1, 1, "AH droite", horizontalalignment="right")  
plt.show()
```





Il existe d'autres modifications possibles du texte, mais celles listées précédemment sont les plus courantes, je vous laisse vous documentez si vous souhaitez en savoir plus.

# Modification des axes

# LIMITES DES AXES X ET Y

- Axe des x

**plt.xlim(xmin = int/float, xmax = int/float)** ou bien **plt.xlim(int/float, int/float)**

- Axe des y

**plt.ylim(ymin = int/float, ymax = int/float)** ou bien **plt.ylim(int/float, int/float)**

xmin et ymin représentent le minimum d'où vos axes doivent partir, et xmax et ymax le maximum d'où vos axes doivent s'arrêter.

- x et y

Vous pouvez également modifier les limites des axes x et y en une ligne avec la commande suivante :

**plt.axis([xmin, xmax, ymin, ymax])**

# EXEMPLE

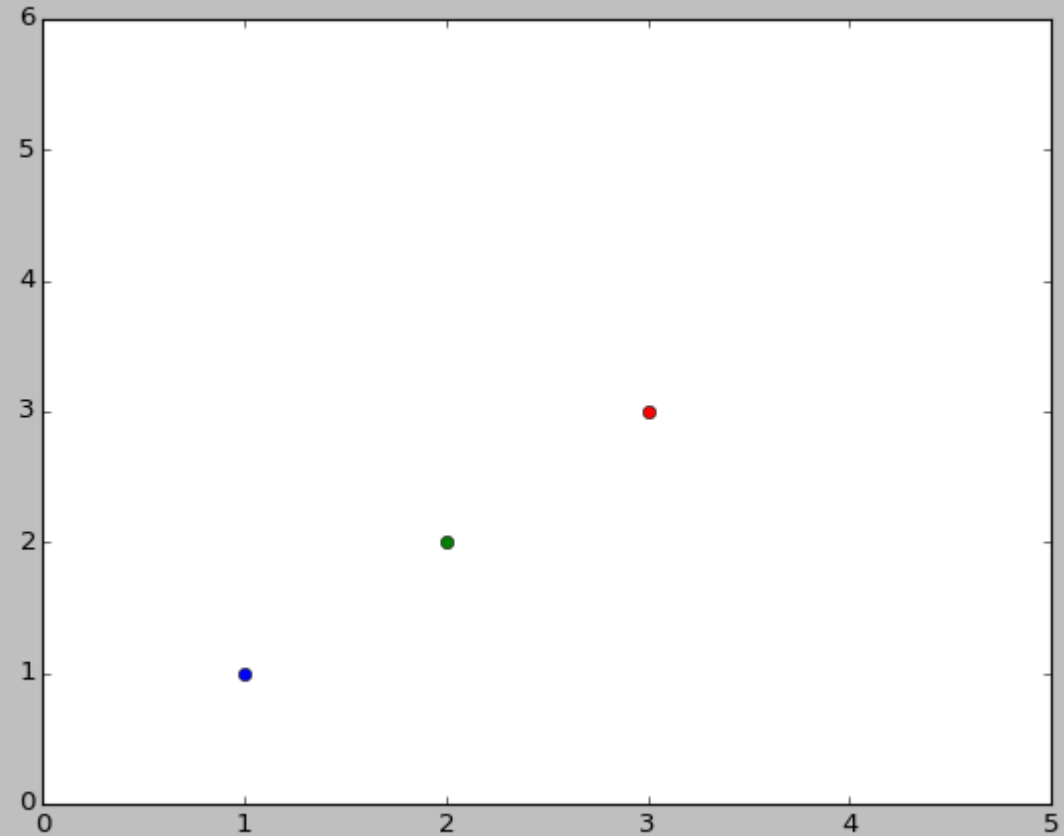
```
plt.plot(1, 1, "o")  
plt.plot(2, 2, "o")  
plt.plot(3, 3, "o")
```

```
plt.xlim(0, 5)  
plt.ylim(ymin=0, ymax=6)
```

```
plt.show()
```

Ou bien

```
plt.axis([0, 5, 0, 6])
```





# CHANGER LA GRADUATION

- Axe des x

**plt.xticks(position, graduation, ...)**

- Axe des y

**plt.yticks(position, graduation, ...)**

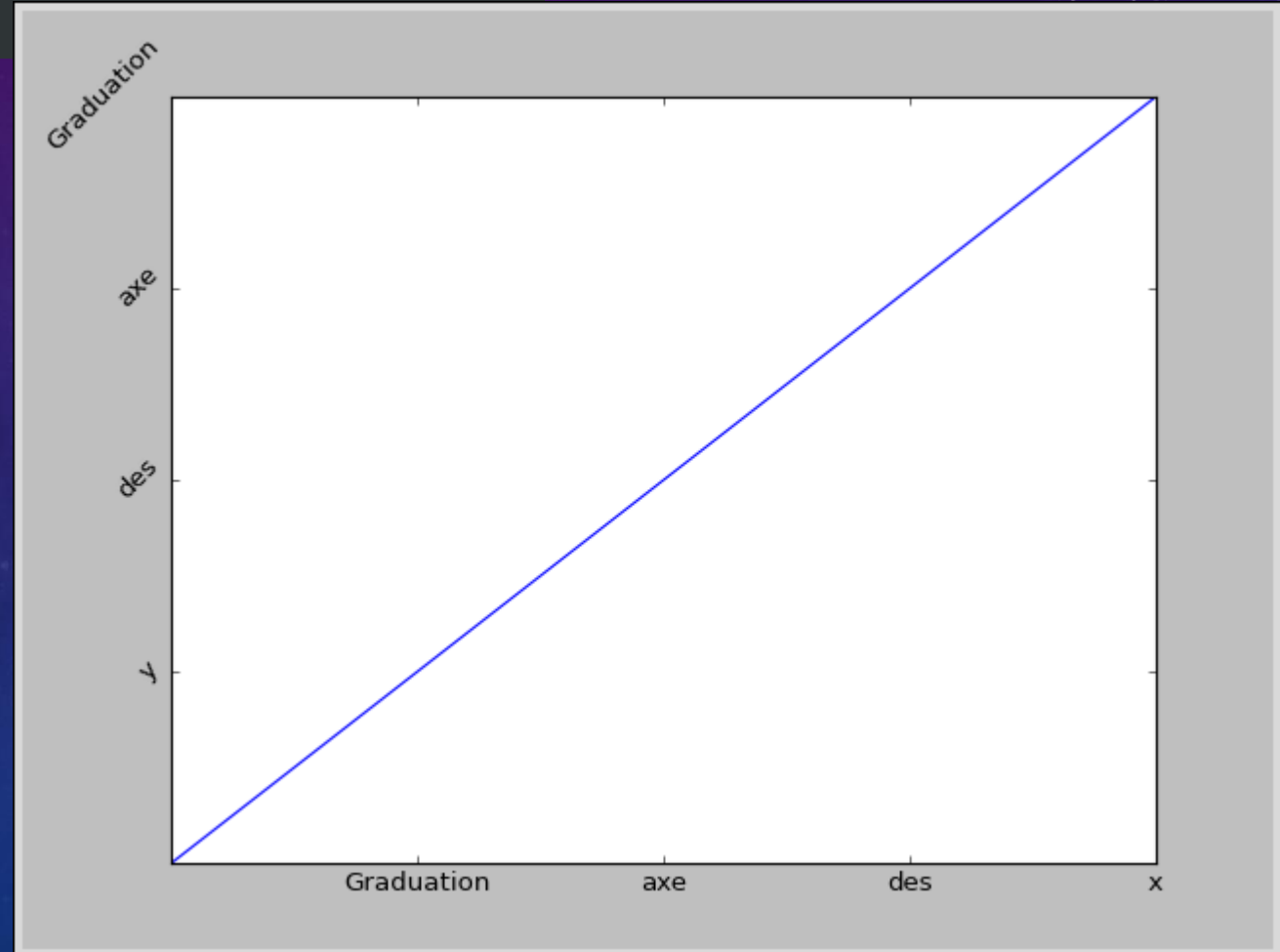
position et graduation sont la plupart du temps (voir tout le temps) des listes et doivent être de même dimension. La liste position spécifie la position de chacune de vos nouvelles graduations contenues dans la liste graduation. L'avantage de pouvoir changer la graduation est que vous pouvez mettre du texte à la place de nombres.

Pour la liste position, on utilise généralement un range() de la longueur de la liste graduation. Vous pouvez choisir de placer votre première graduation à 0 ou un 1. Mais une petite manipulation de votre range() s'impose. Si vous ne savez pas comment, revoyez votre cours sur la manipulation des listes et du range().

Il est possible de passer des arguments à ces fonctions. Ces arguments sont les mêmes que pour du texte (couleur, taille, orientation, alignement, ...).

# EXAMPLE

```
plt.plot(range(5), range(5))  
plt.xticks(range(1, 5, 1), ["Graduation", "axe", "des", "x"])  
plt.yticks(range(4, 0, -1), ["Graduation", "axe", "des", "y"], rotation=45)  
  
plt.show()
```



# ENREGISTRER L'IMAGE



Vous pouvez enregistrer votre graphe, soit manuellement en cliquant sur l'icône enregistrer en bas de la fenêtre de votre graphe, ou avec la commande **plt.savefig(« nom »)**. N'oubliez pas de préciser l'extension (.png, .bmp, .jpg, ...). Par défaut votre graphe sera enregistré dans le répertoire de votre script, mais vous pouvez spécifier le chemin du dossier où vous voulez enregistrer votre figure « home/Documents/.../nom.png ».

# Autres éléments utiles



# Ajouter une ligne horizontale et/ou verticale

- Ligne verticale

**plt.axvline(x = int/float, ymin = 0, ymax = 1, ...)**

**plt.axvline(x, ymin, ymax, ...)**

**plt.axvline(x, ...)**

- Ligne horizontale

**plt.axhline(y = int/float, xmin = 0, xmax = 1, ...)**

**plt.axhline(y, xmin, xmax, ...)**

**plt.axhline(y, ...)**

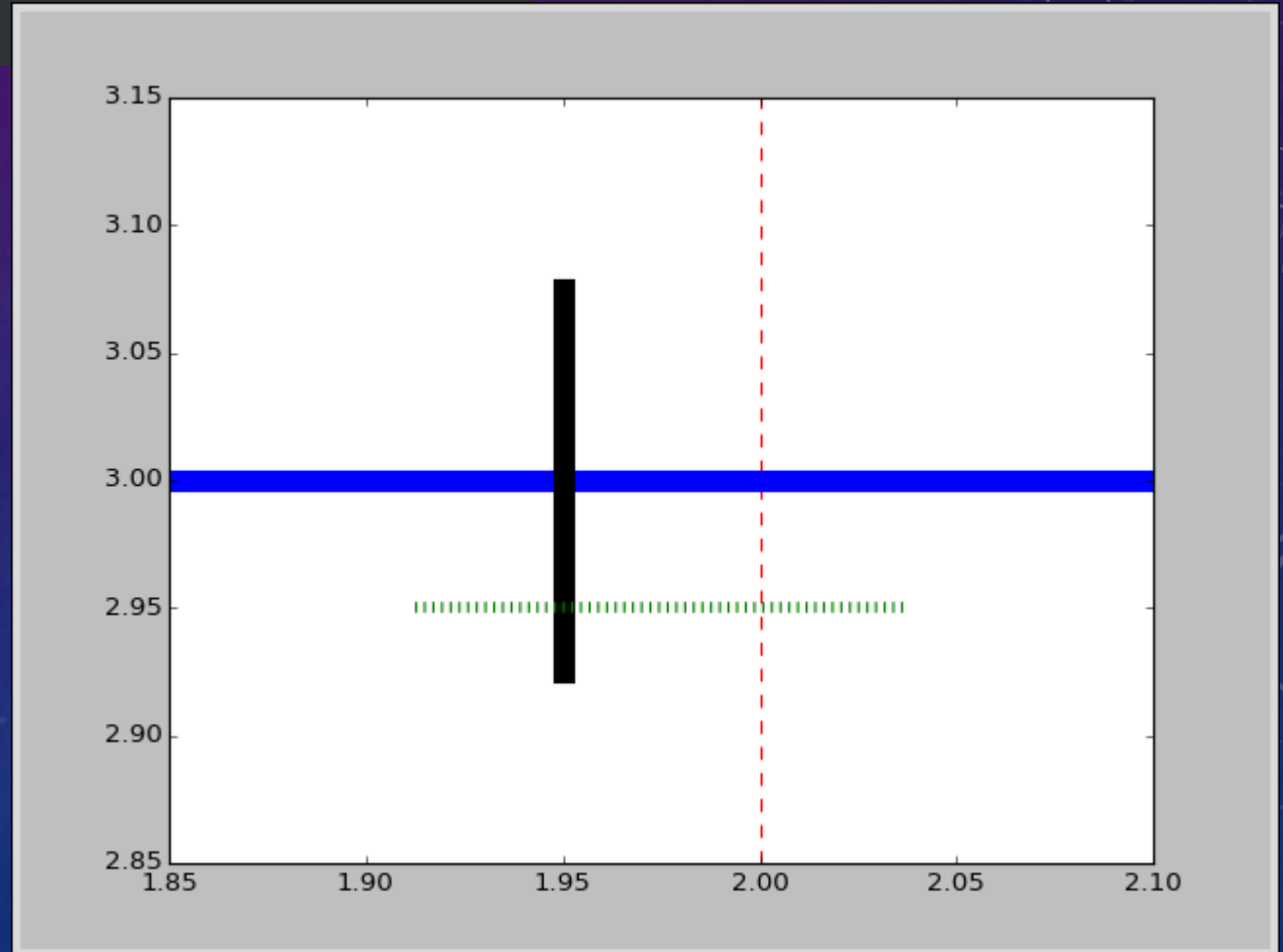
Pour la ligne verticale, par défaut, les valeurs de ymin et ymax sont fixées à 0 et 1 (donc y doit être compris entre 0 et 1). Voyez cela en pourcentage par rapport aux dimensions du graphe. 0 représente le bas du graphe, et 1 le haut du graphe. Si vous insérez 0 à ymin et 0,75 à ymax, votre ligne commencera du bas du graphe et terminera aux  $\frac{3}{4}$  de la largeur du graphe.

Même chose pour la ligne horizontale. 0 représente le bord gauche, et 1 le bord droit.



# EXAMPLE

```
plt.axvline(2, c='r', linestyle="--")  
plt.axhline(3, lw=10)  
plt.axvline(x=1.95, ymin=0.25, ymax=0.75, c="k", lw=10)  
plt.axhline(y=2.95, xmin=0.25, xmax=0.75, c="g", linestyle=":", lw=5)  
  
plt.show()
```



# Ajouter un rectangle horizontal et/ou vertical

- Rectangle horizontal

`axhspan(ymin = int/float, ymax = int/float, xmin = 0, xmax = 1, ...)`

`axhspan(ymin, ymax, ...)`

- Rectangle vertical

`axvspan(xmin = int/float, xmax = int/float, ymin = 0, ymax = 1, ...)`

`axvspan(xmin, xmax, ...)`

Trace un rectangle de `ymin` à `ymax` pour un rectangle horizontal et sur toute la longueur des `x`. Même principe pour le rectangle vertical. (Les modifications de l'étendu du rectangle est la même que pour les lignes, voir diapo 66).

Par défaut le rectangle est rempli (**fill = True**), mais vous pouvez ne pas le remplir avec **fill = False**.

Si vous placez plusieurs rectangles sur un même graphe, ils se superposent selon l'ordre dans lequel vous les placerez.

Les modifications de l'épaisseur des lignes, couleur, style de ligne restent les mêmes.

# QUELQUES ARGUMENTS

- Opacité - **alpha** = int/float

invisible <- opacité -> opaque		
0	...	1

- Hachurage - **hatch** = « »

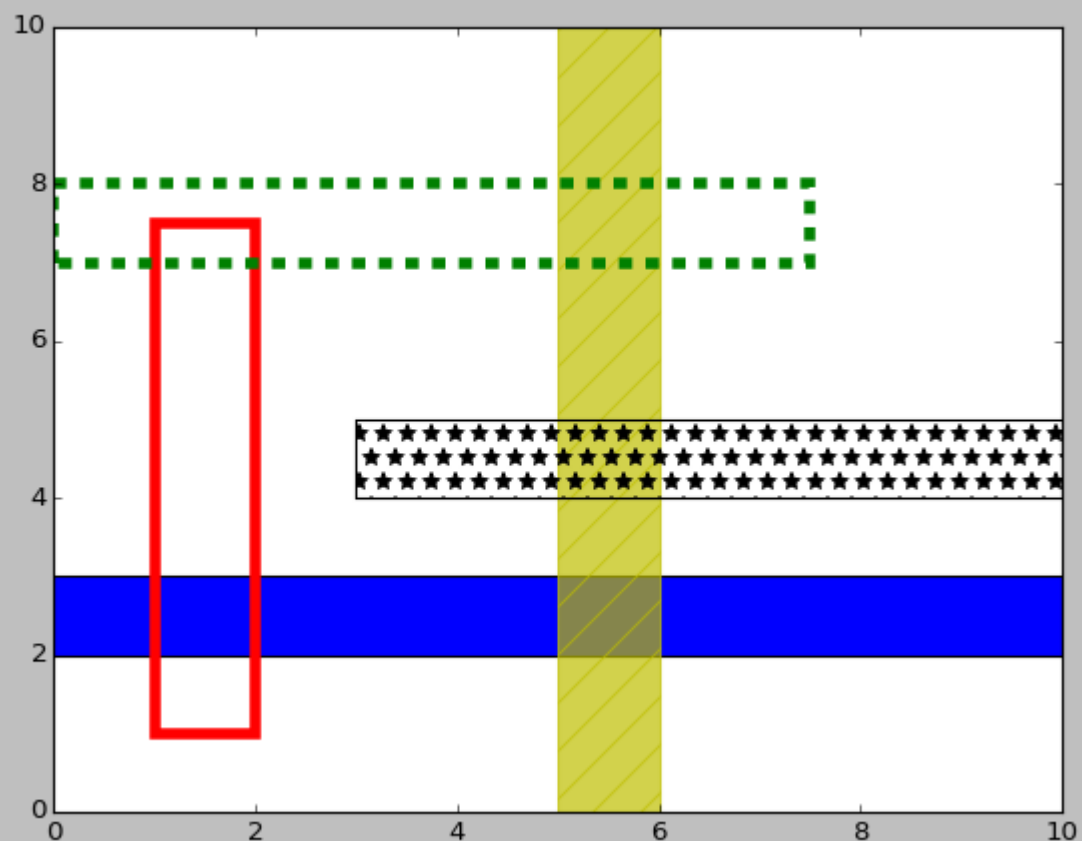
/	\		-	+	x	o	O	.	*
---	---	--	---	---	---	---	---	---	---

```
plt.axis([0, 10, 0, 10])

plt.axhspan(5, 6, alpha=0.7, color='y', hatch='/')
plt.axvspan(1, 2, 0.10, 0.75, fill=False, color='r', lw=5)
plt.axhspan(4, 5, 0.30, 1, fill=False, color='k', hatch='*')
plt.axhspan(7, 8, 0, 0.75, linestyle='dashed', fill=False, color='green', lw=5)

plt.show()
```

## EXAMPLE



# GRILLE

## `plt.grid(True/False, ...)`

Ajoute la grille (lignes verticales et horizontales) sur votre graphe. Par défaut, elle n'est pas affichée.

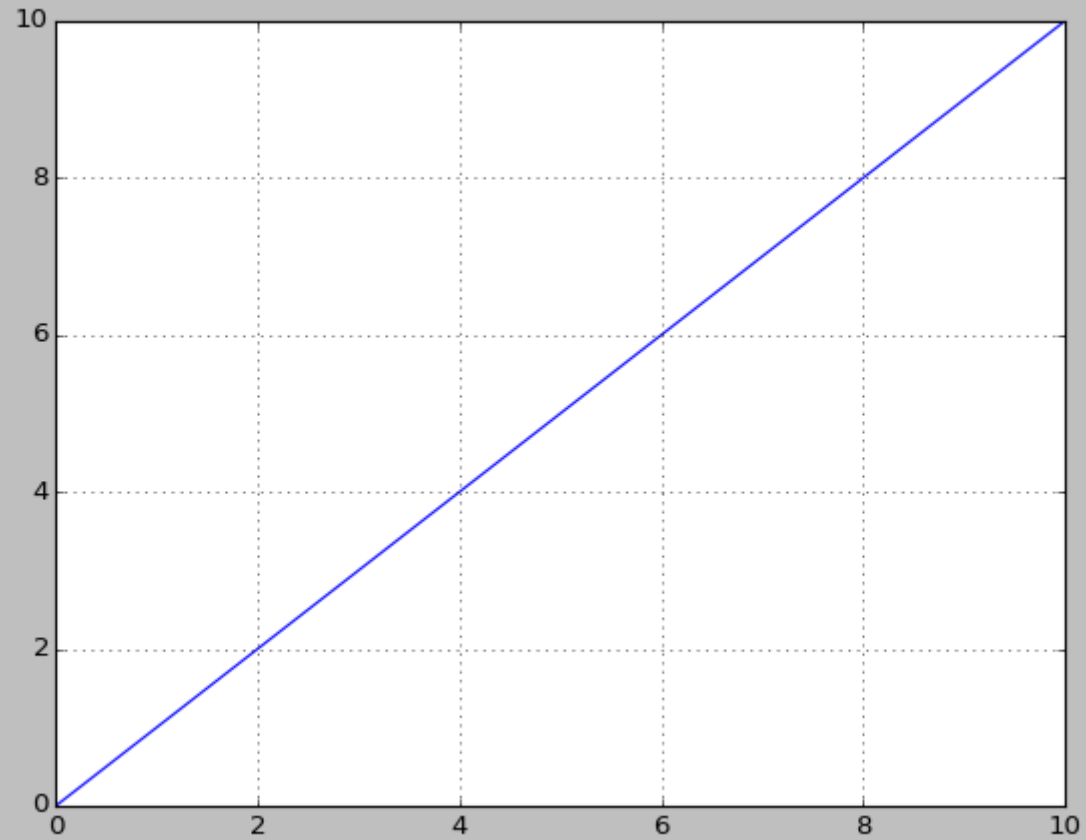
Si True, les lignes sont affichées dans les deux orientations (**axis = 'both'**). Vous pouvez choisir lesquelles afficher avec **axis = 'x'** pour les lignes verticales ou **axis = 'y'** pour les lignes horizontales.

Vous pouvez ajouter les arguments de modification des lignes (couleur, épaisseur, style, ...).

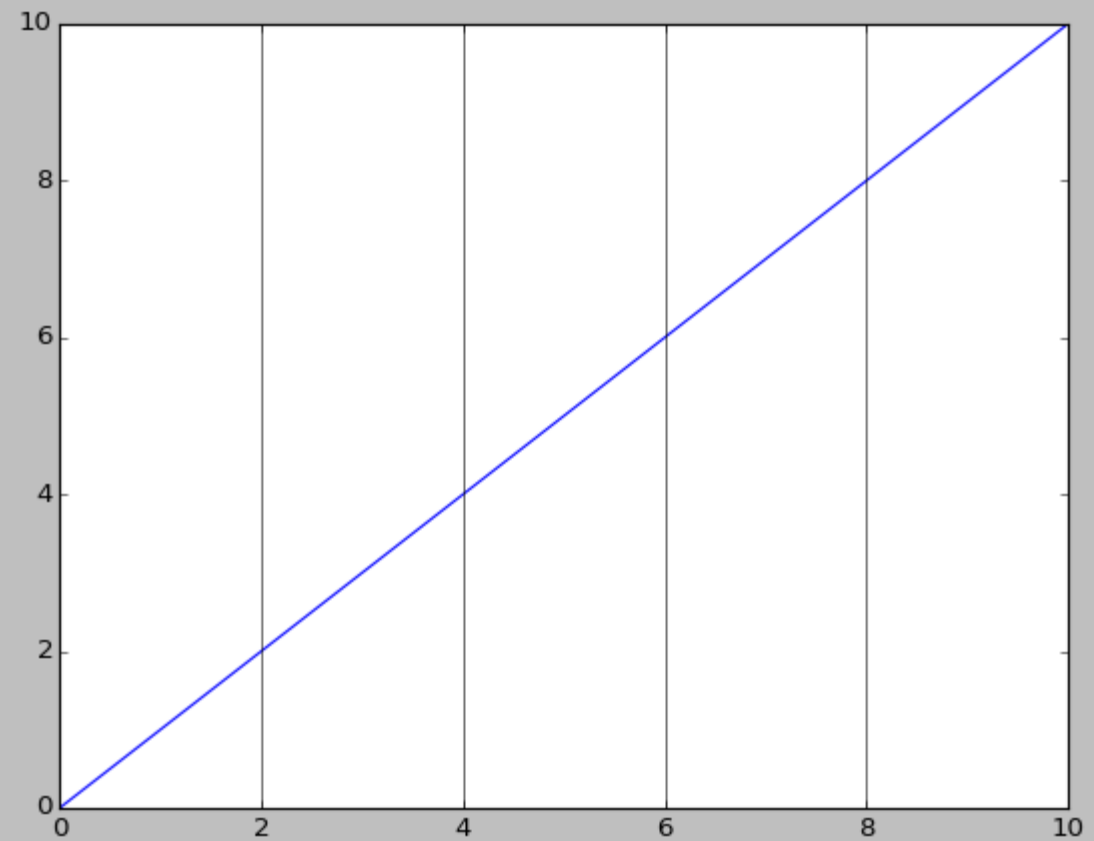


# EXAMPLE

```
plt.axis([0, 10, 0, 10])  
plt.plot([0, 10], [0, 10])  
plt.grid(True)  
plt.show()
```



```
plt.axis([0, 10, 0, 10])  
plt.plot([0, 10], [0, 10])  
plt.grid(True, axis='x', linestyle='-')  
plt.show()
```



# Courbes et fonctions mathématiques

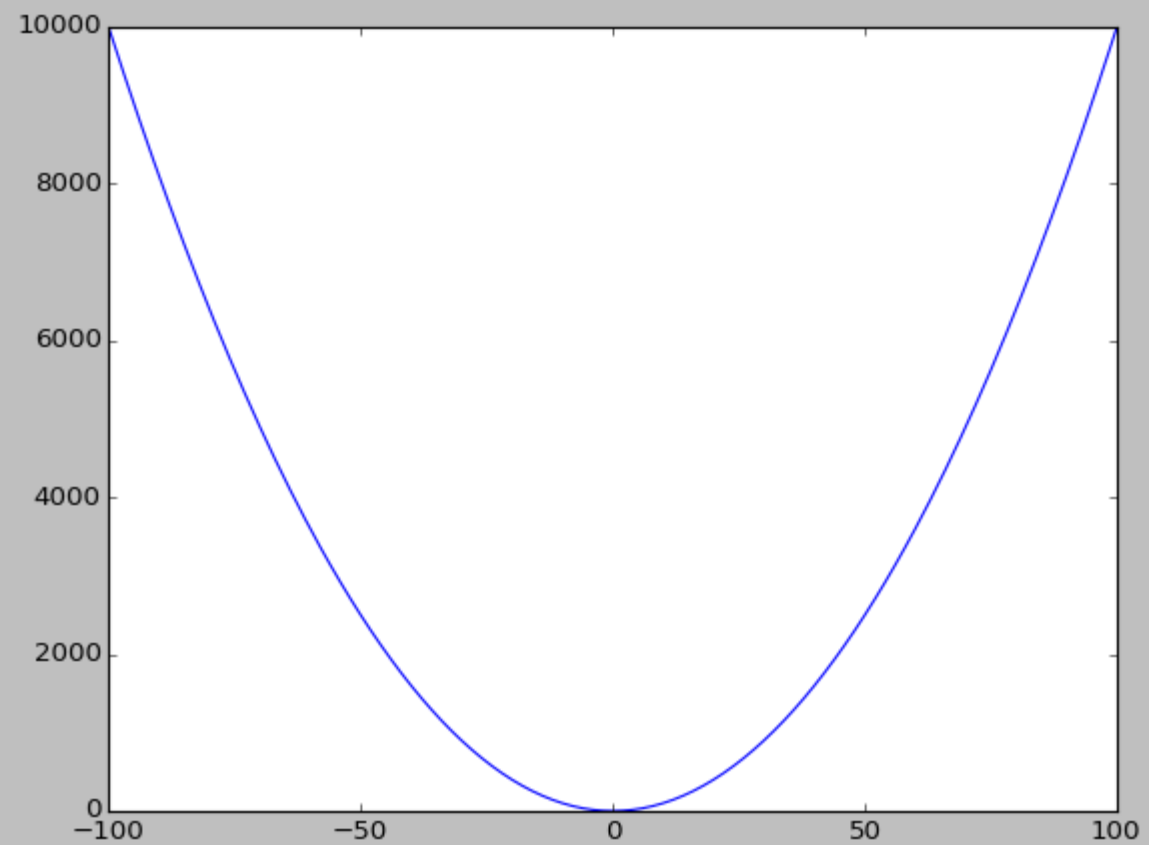
The background is a gradient of dark blue to purple, speckled with small white dots. It features several faint, light blue mathematical diagrams. In the top right, there is a large circular scale with degree markings from 0 to 210 and concentric circles with arrows indicating rotation. In the bottom right, there are concentric circles with arrows. In the bottom left, there is a partial circular diagram with an arrow. In the top left, there is a small circular diagram with an arrow.

# TRACER DES FONCTIONS MATHÉMATIQUES

Il est possible de tracer des fonctions mathématiques simplement. Dans l'exemple qui suit, la fonction  $f(x) = x^2$  est tracée. Mais vous pouvez utiliser n'importe quel autre fonction.

```
x = range(-100, 101, 1)
y = [i**2 for i in x]

plt.plot(x, y)
plt.show()
```





# Diagrammes en barres

The background is a gradient of dark blue and purple, speckled with white dots resembling a starry sky. Overlaid on this are several faint, white technical diagrams. In the top right, there is a large circular gauge with concentric circles and radial tick marks, with numbers 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, and 200 visible along its outer edge. In the bottom right, there is a diagram of two concentric circles with dashed lines and arrows indicating a clockwise flow. In the bottom left, there is a partial view of a similar circular diagram with arrows. In the top left, there is a small circular diagram with a single arrow.

# AVANT DE COMMENCER

Il faut savoir que les diagrammes en barres (que vous allez découvrir tout de suite) sont différents des histogrammes (que vous allez découvrir pas tout suite), qui sont également des diagrammes ... en barres ! En effet, leur différence réside dans les données. Vous allez comprendre en lisant les diapos qui suivent.

# C'EST QUOI ??????????

C'est un graphe avec ... Des barres ! Ou si vous voulez, des bâtons ! Mais encore, des rectangles ! Mais encore ... bon ça suffit. Vous l'aurez compris, un diagramme en barres (à barres ?). Utilisé généralement pour visualiser l'effectif de plusieurs catégories, blablabla ... vous savez à quoi ça sert. Je passe les détails et allons directement à la pratique !

Pour votre diagramme à barres vous avez besoin ... d'une liste de valeurs, ainsi que d'une liste « position » des barres. La première liste contient des entiers et/ou des réelles qui représenteront la hauteur de vos barres (notez qu'il serait inutile d'afficher qu'une seule barre, c'est pourquoi, c'est une liste !). La deuxième liste contient la position de chacune de vos barres selon l'axe des x (pour des barres verticales), ou selon l'axe des y (pour des barres horizontales). Ces deux liste doivent être de même dimension.

Afin de créer votre diagramme, la syntaxe est la suivante :

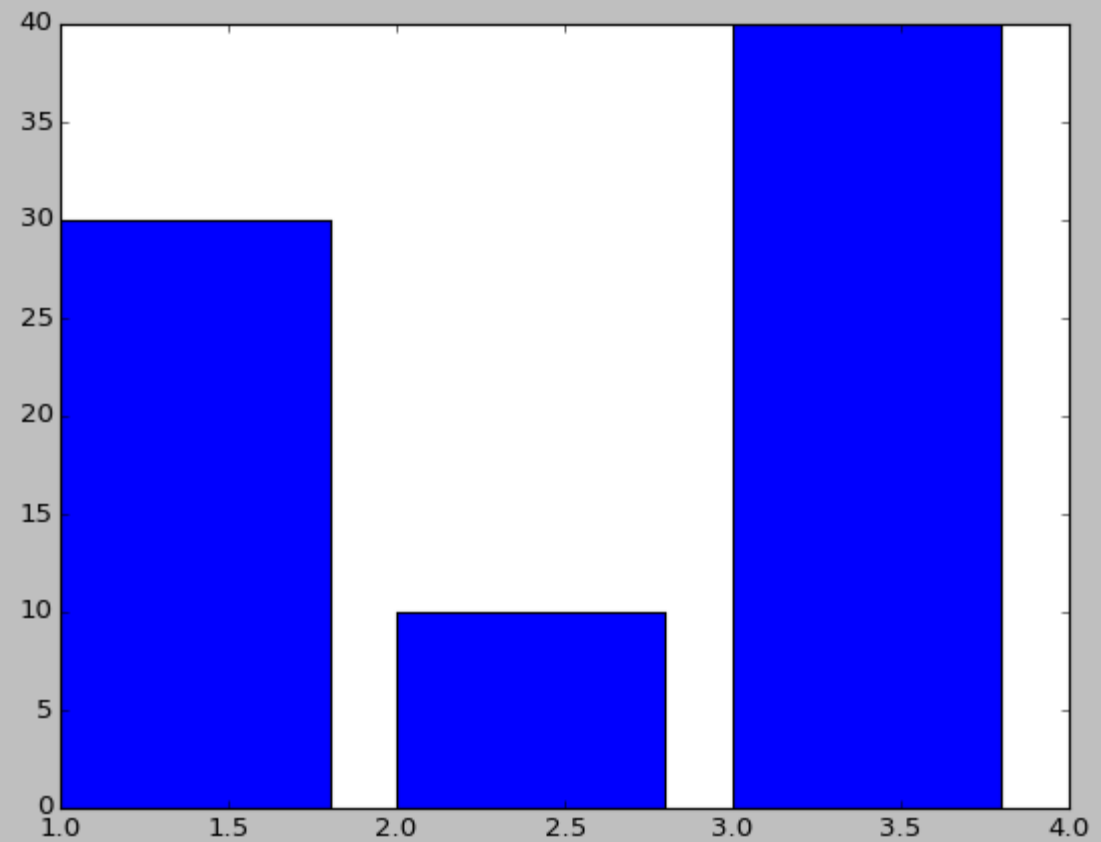
**plt.bar(position, valeurs, ...)**

Pour la liste « position », en général, un range() du nombre de valeurs de la liste qui contient les valeurs est souvent utilisé. Le range commençant souvent à 0 si vous n'en précisez pas le début, vous pouvez le faire commencer à 1, mais n'oublier de rajouter une dernière valeur, car pour 6 valeurs pour range(6) par exemple, le 6 est exclu (range(1, 7) par exemple). Libre à vous de choisir vous-même la position des barres.

Mais voyons un exemple tout de suite.



```
plt.bar([1, 2, 3], [30, 10, 40])  
plt.show()
```





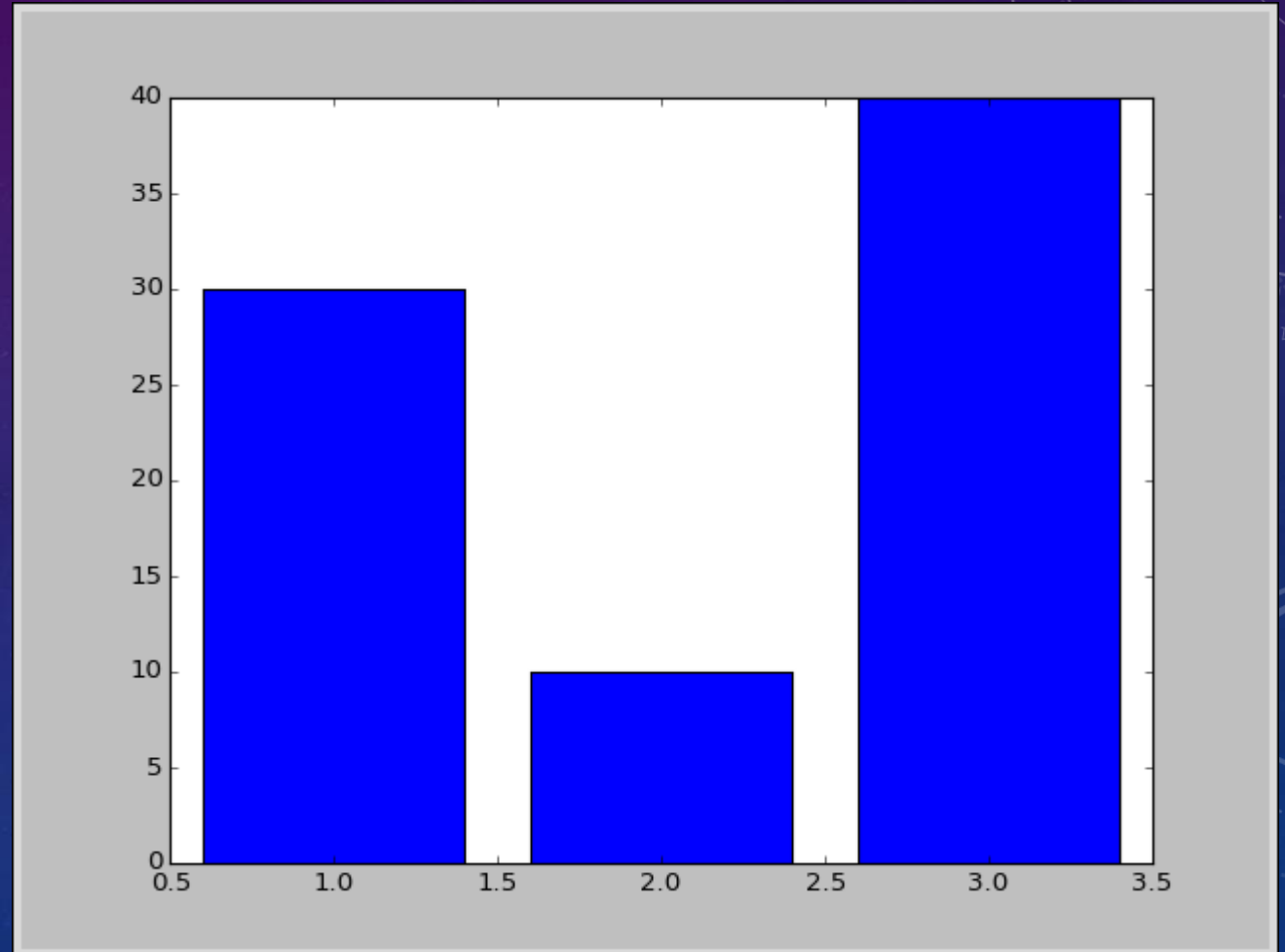
La liste position donne en fait la position du bord gauche de chaque barre. Il est bien sûr possible de modifier cela en le précisant en argument à la suite de la liste de vos valeurs dans **plt.bar()**. D'autres modifications sont également possibles, mais voyons les tout de suite !

# ALIGNEMENT DES BARRES (AXE DES X)

Par défaut, le bord gauche (edge) de chaque barre est sur la position donnée par la liste position. Il est possible centrer les barres sur la position en utilisant **align = « center »** (**align = « edge »** par défaut).

# EXAMPLE

```
plt.bar([1, 2, 3], [30, 10, 40], align='center')  
plt.show()
```

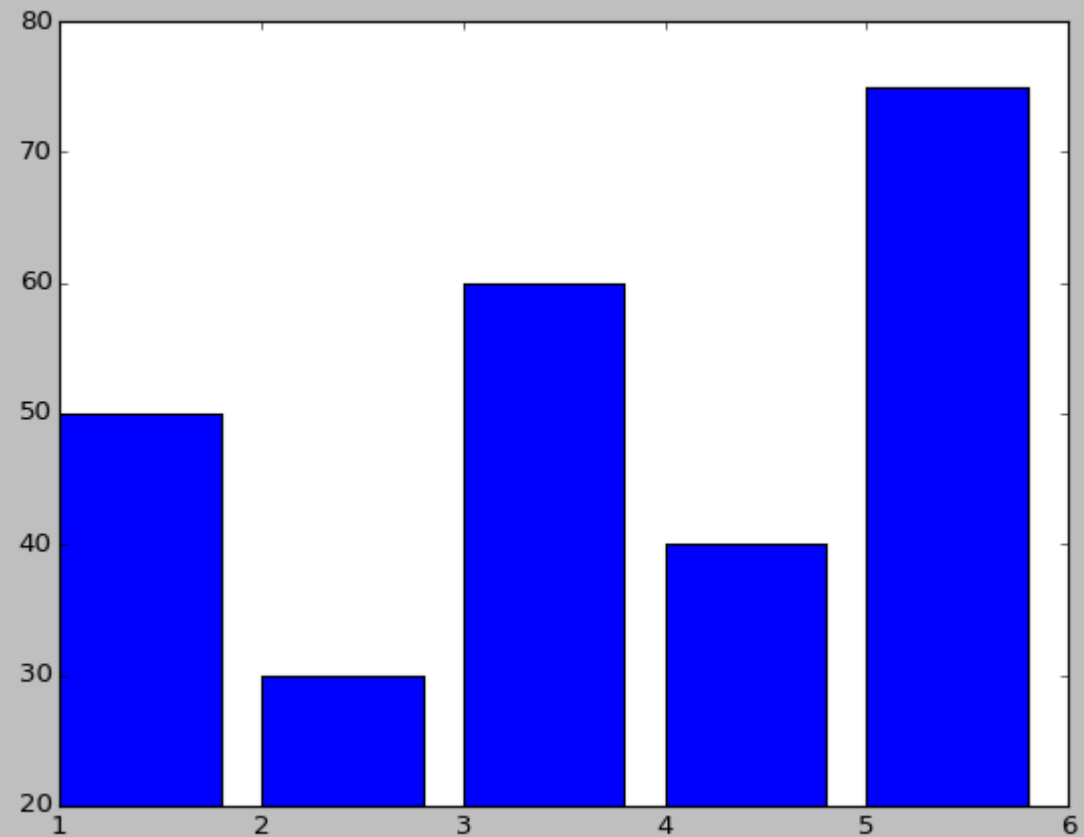


## ALIGNEMENT DES BARRES (AXE DES Y)

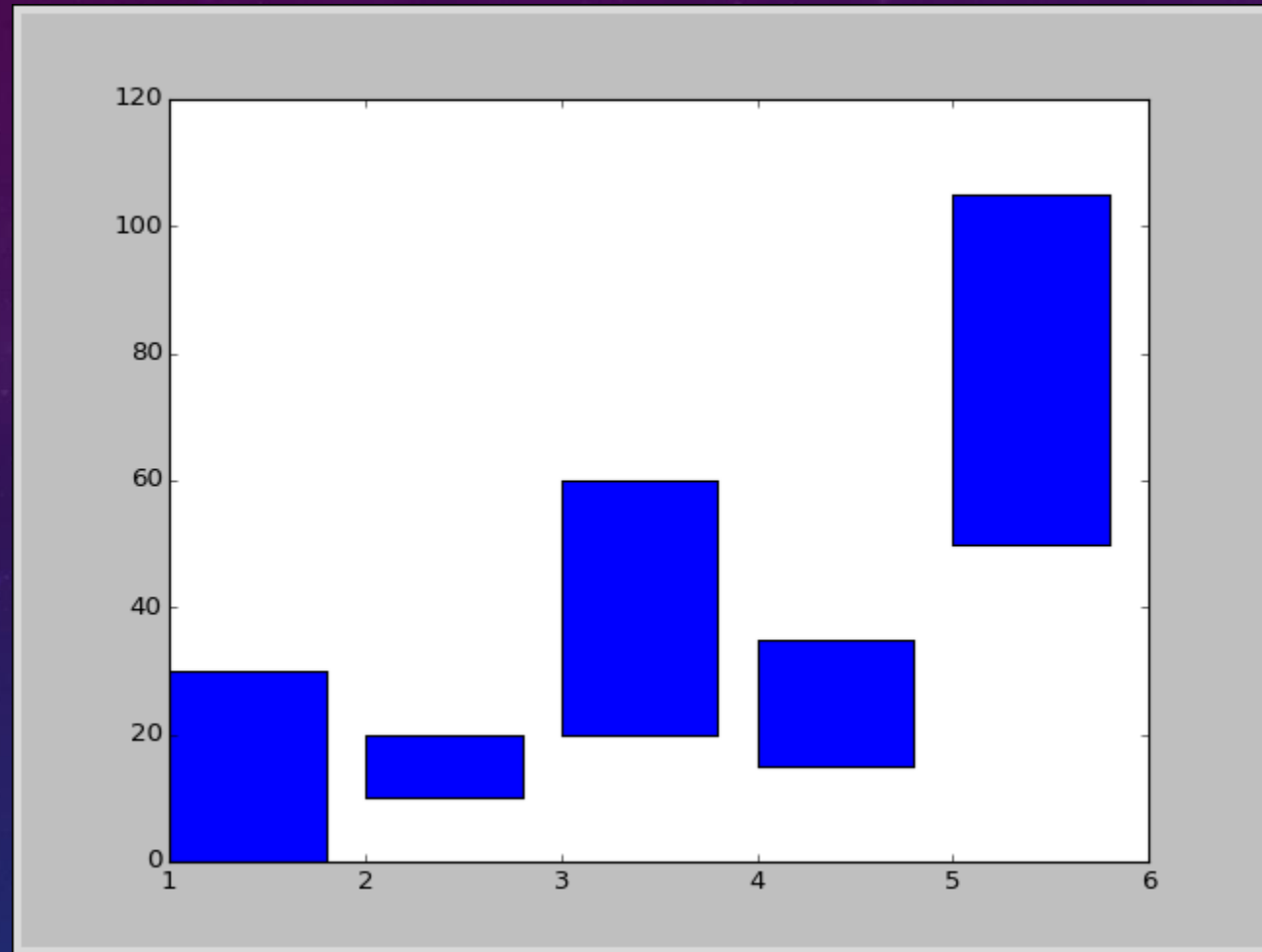
Il est également possible de modifier la position de vos barres selon l'axe des y. Pour cela, utiliser **bottom = int/float**, avec int/float une valeur qui détermine la position de la base des barres. Si une valeur unique est entrée, toutes les barres seront affectées de la même façon. Vous pouvez sinon donner une liste d'autant de valeur que de barres. Chaque valeur donnant la position de la base de chaque barre selon l'axe des y. Par défaut, la base des barres est à 0.

# EXAMPLE

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], bottom=20)  
plt.show()
```







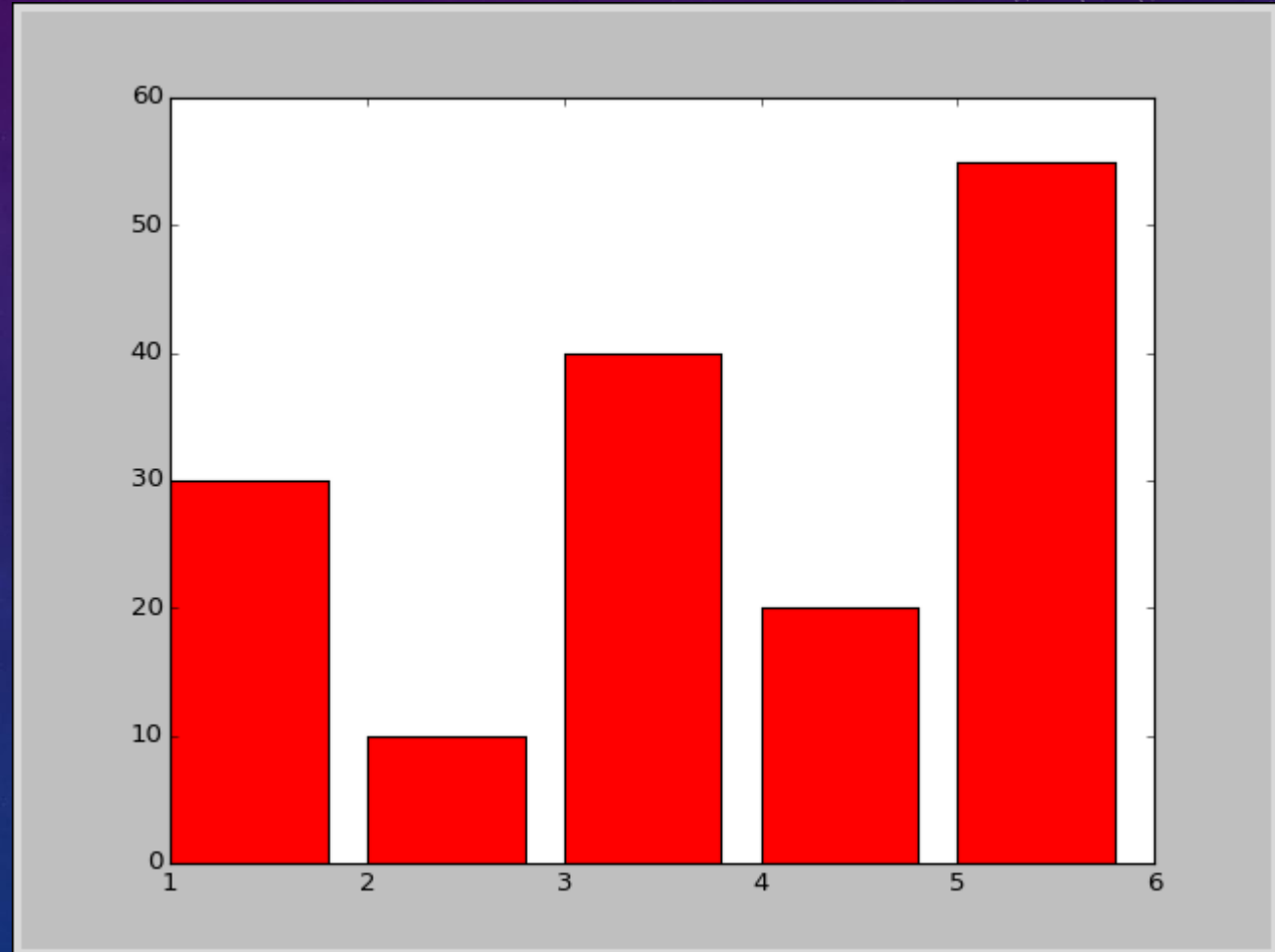
```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], bottom=[0, 10, 20, 15, 50])  
plt.show()
```

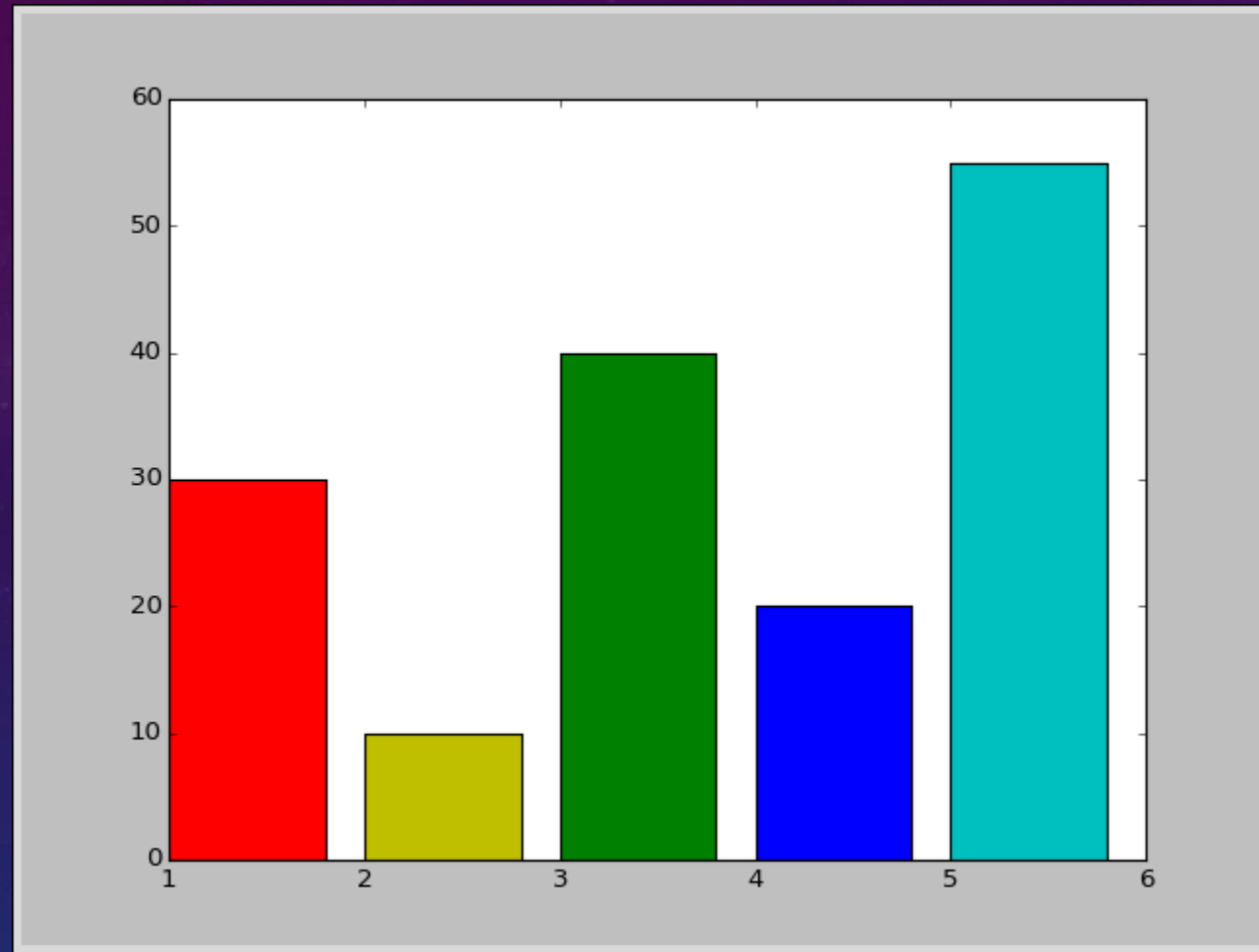
# COULEUR

Avec **color** = « ». Même principe qu'avec **bottom** = « ». Soit vous insérez une couleur unique, soit une liste de couleurs. Les couleurs sont les mêmes que pour les courbes (diapo 39 et 42).

# EXAMPLE

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], color='r')  
plt.show()
```





```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], color=['r', 'y', 'g', 'b', 'c'])  
plt.show()
```

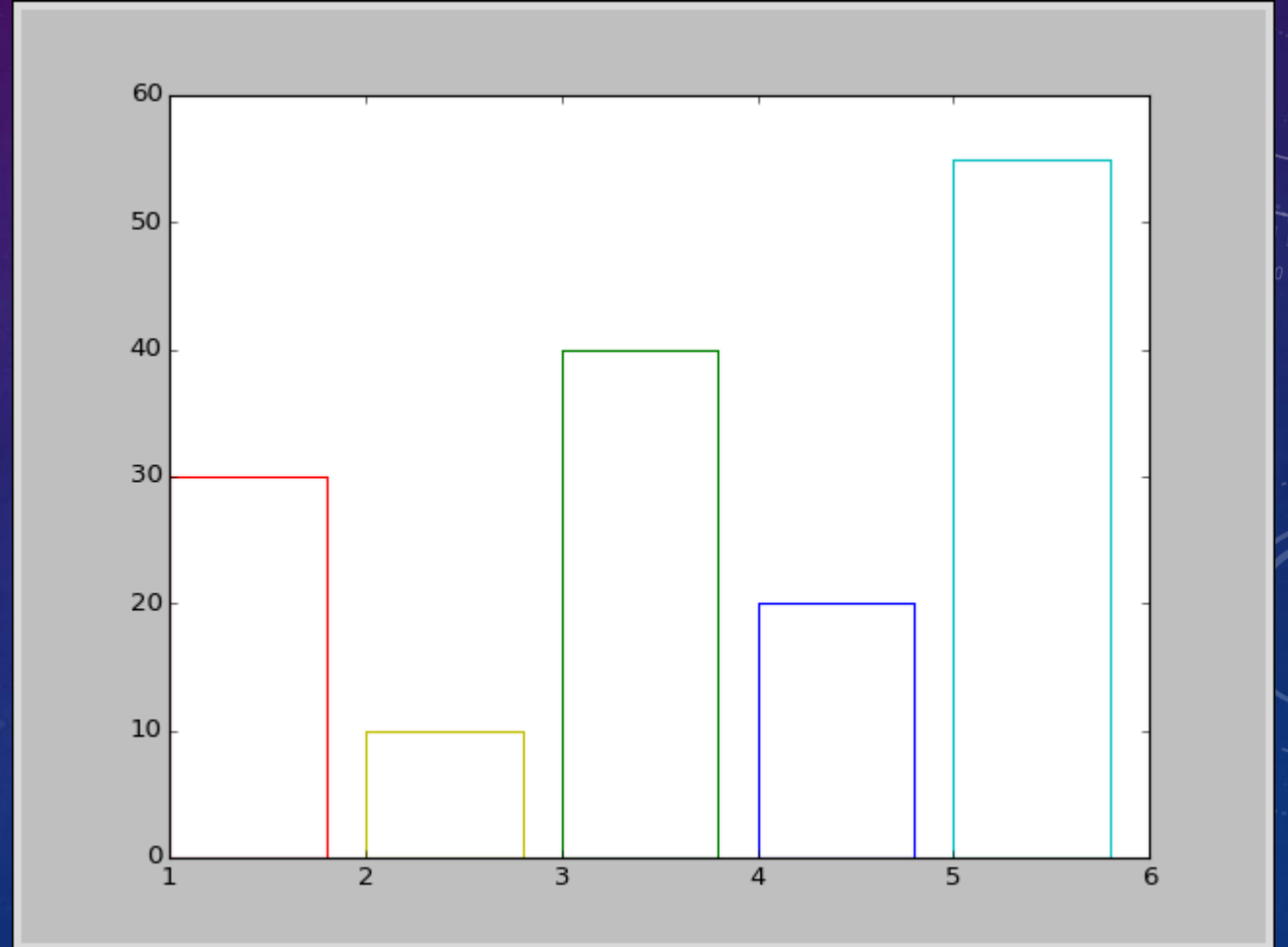
# COULEUR DES BORDS DES BARRES

Même chose, mais avec **edgecolor** = « **string** ».  
Couleur noire par défaut.



# EXAMPLE

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], color="w", edgecolor=['r', 'y', 'g', 'b', 'c'])  
plt.show()
```



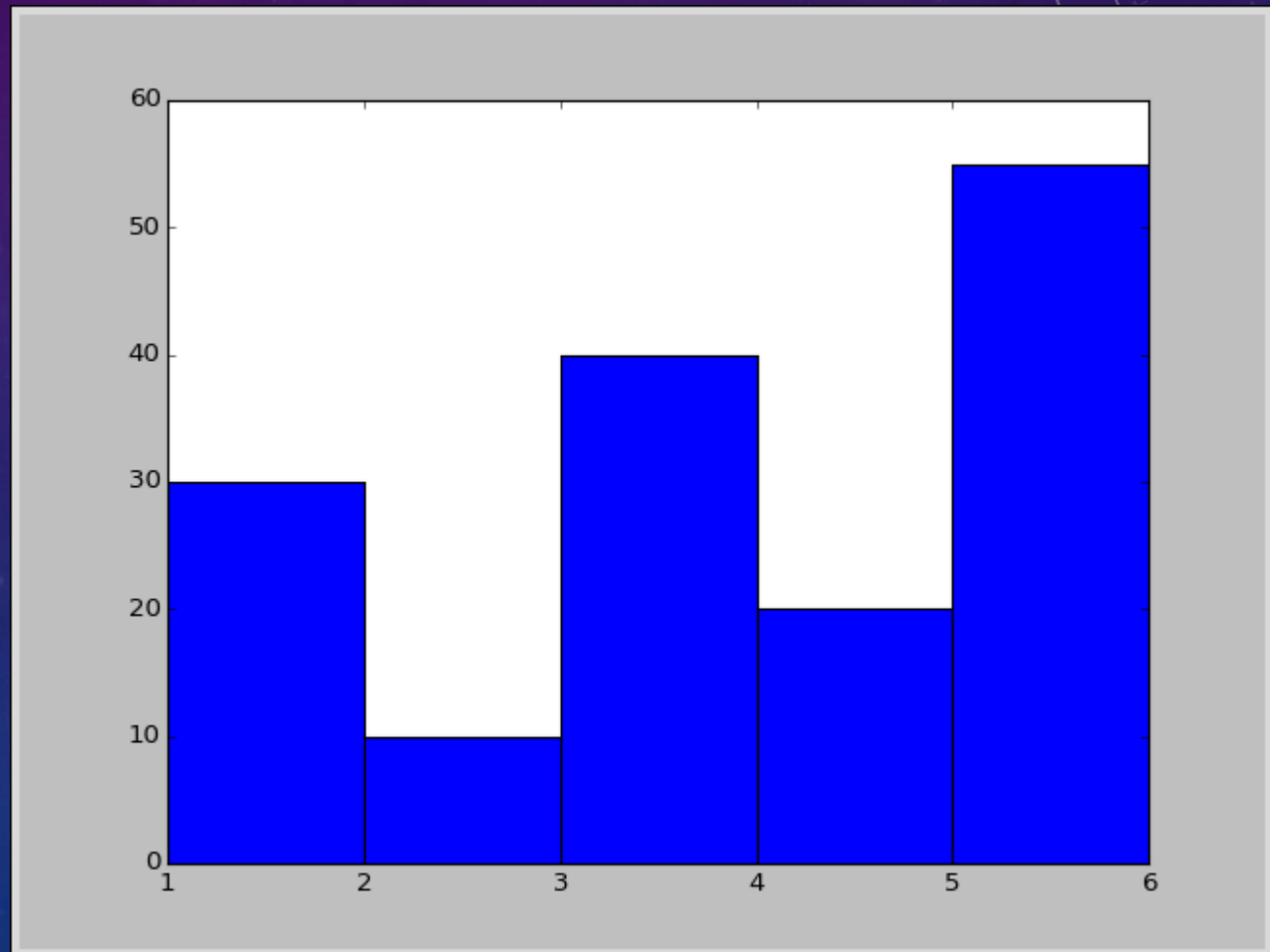
# LARGEUR DES BARRES

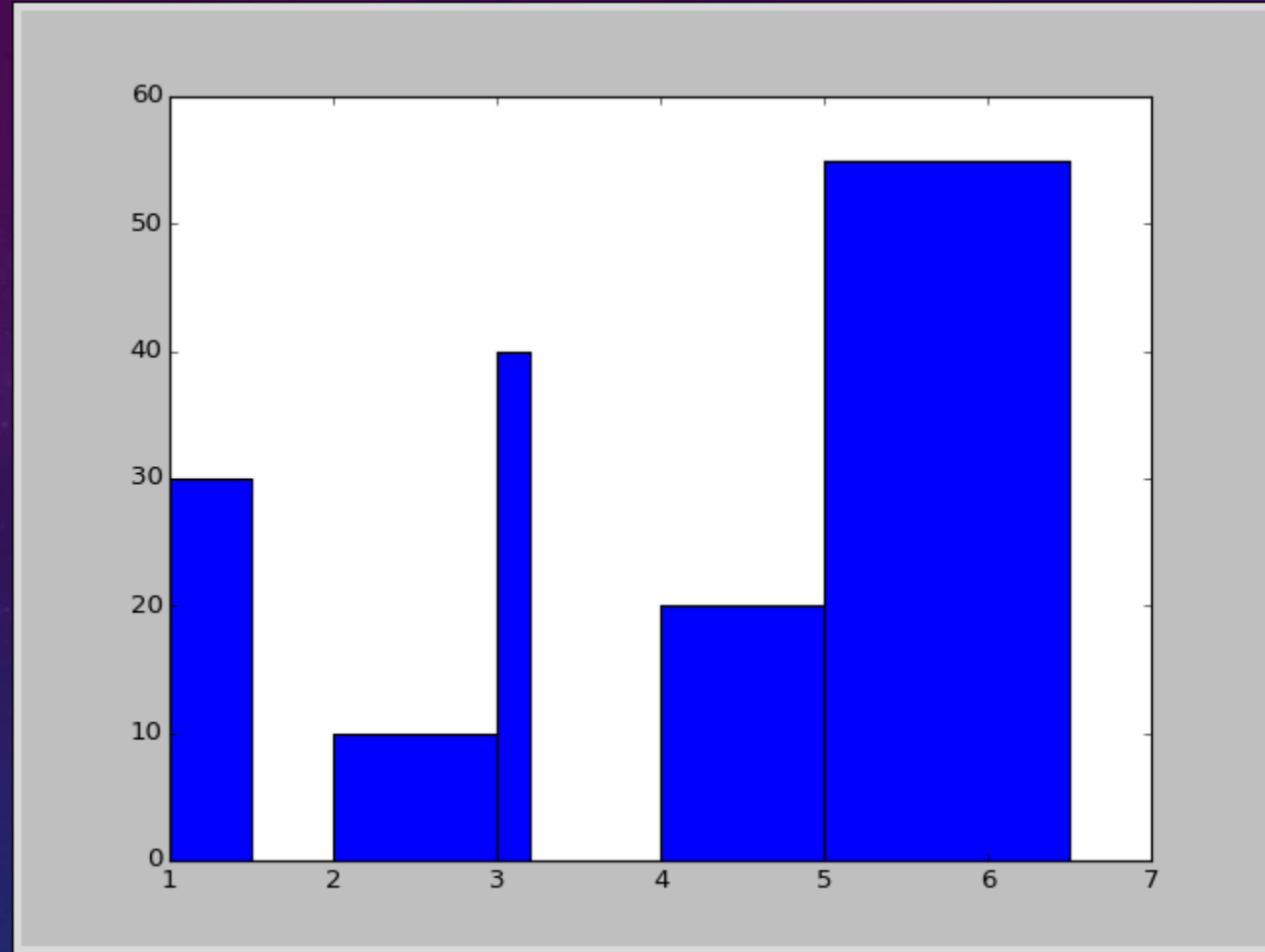
Encoooooooooooooooooore la même chose, avec **width = int/float**. Soit vous entrez une valeur unique et toutes vos barres seront de la largeur précisée, soit une liste d'autant de valeurs que de barres.

Par défaut, la largeur des barres est de 0,8.

# EXEEEEEEEEEEEEEEEEEMPLE

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], width=1)  
plt.show()
```





```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], width=[0.5, 1, 0.2, 1, 1.5])  
plt.show()
```

# ÉPAISSEUR DES BORDS (LIGNES) DES BARRES

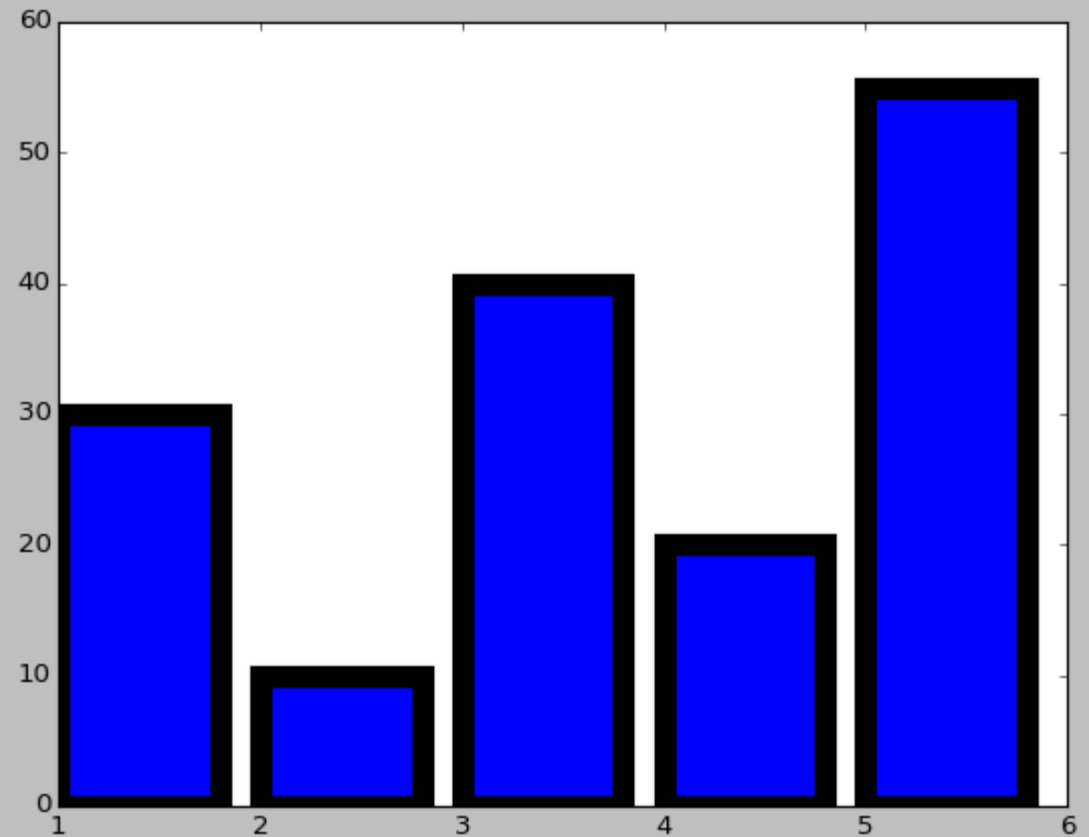
Vous avez sûrement encoooooooooore deviné ... c'est la même chose qu'avec la couleur, la taille, ... avec **linewidth = int/float**. Une valeur unique ou une liste de valeurs, toujours autant de valeurs que de barres. Valeur par défaut : 1.

Si 0, pas de bord tracé.

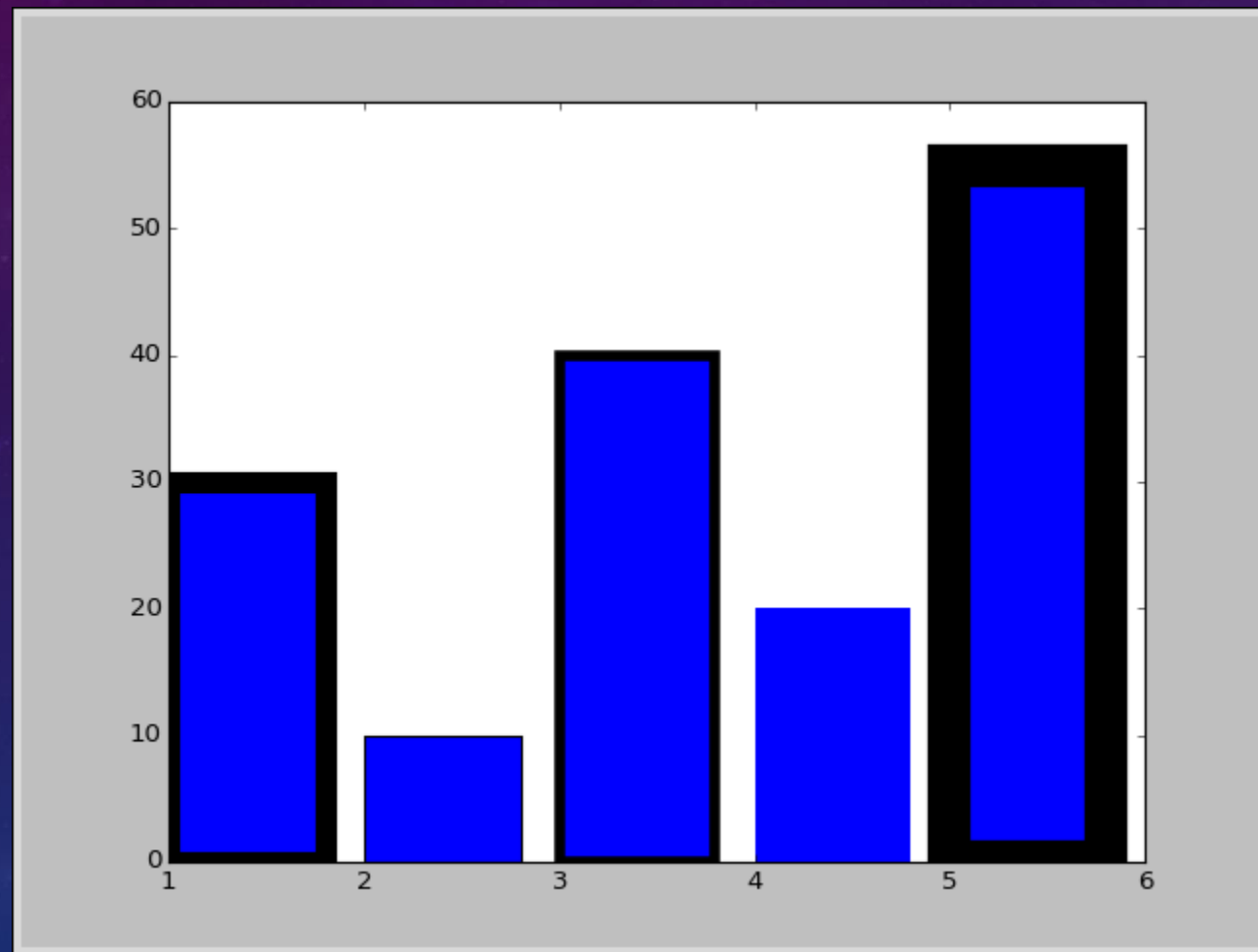


# EXAMPLE ...

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], linewidth=10)  
plt.show()
```



```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], linewidth=[10, 1, 5, 0, 20])  
plt.show()
```



# BARRES D'ERREURS EN X ET Y

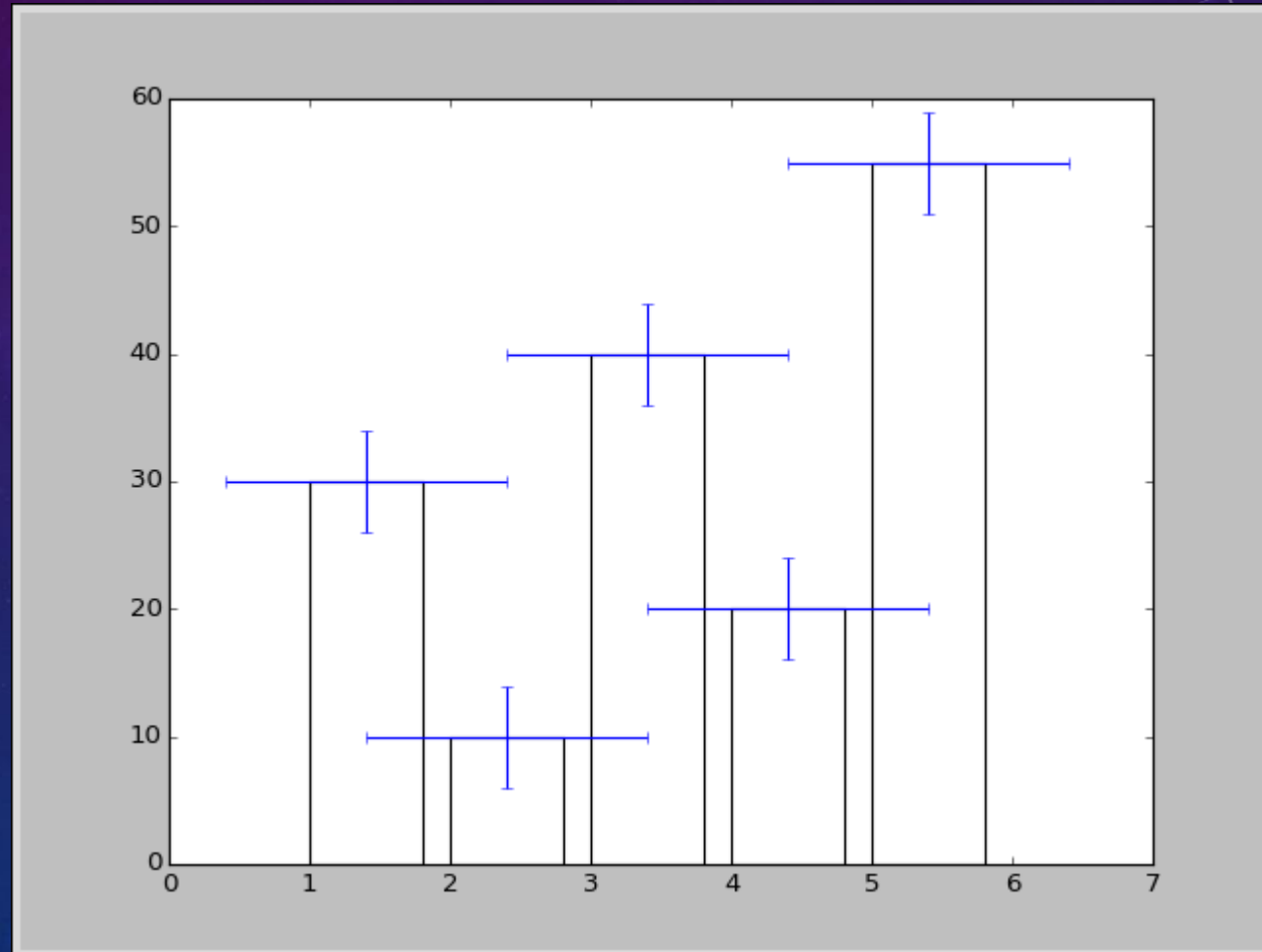
Même principe qu'avec les précédents ...

Utilisez **xerr = int/float** pour une barre d'erreur parallèle à l'axe des x et **yerr = int/float** pour une barre d'erreur parallèle à l'axe des y. Valeur unique ou liste de valeurs.

Ne sont pas affichées par défaut.

# EXAMPLE

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], color="w", xerr=1, yerr=4)  
plt.show()
```



# COULEUR ET TAILLE DES BARRES D'ERREURS

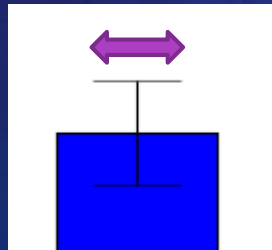
- Couleur

**ecolor** = « »

Une couleur unique ou une liste de couleurs.

- Taille des extrémités des barres d'erreur

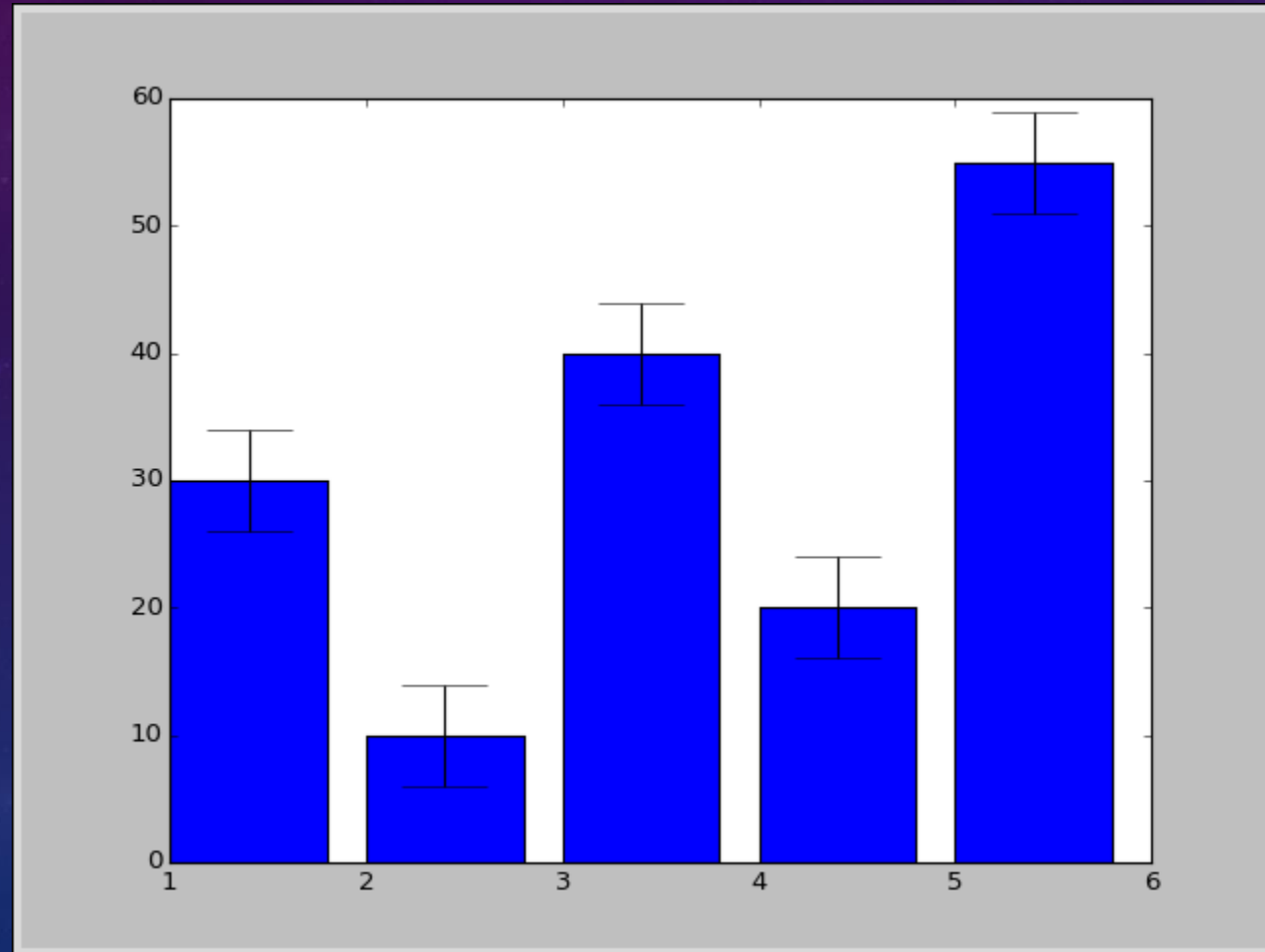
**capsize** = int/float





# ILLUSTRATION

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], yerr=4, ecolor="k", capsize=20)  
plt.show()
```



# LÉGENDES

Voir diapos 51 à 56.

# AFFICHER LA HAUTEUR DES BARRES SUR LE GRAPHE

Si vous vous rappelez l'histoire d'afficher les coordonnées de chaque point à côté du point dans la partie courbes, il est également possible de le faire pour les barres.

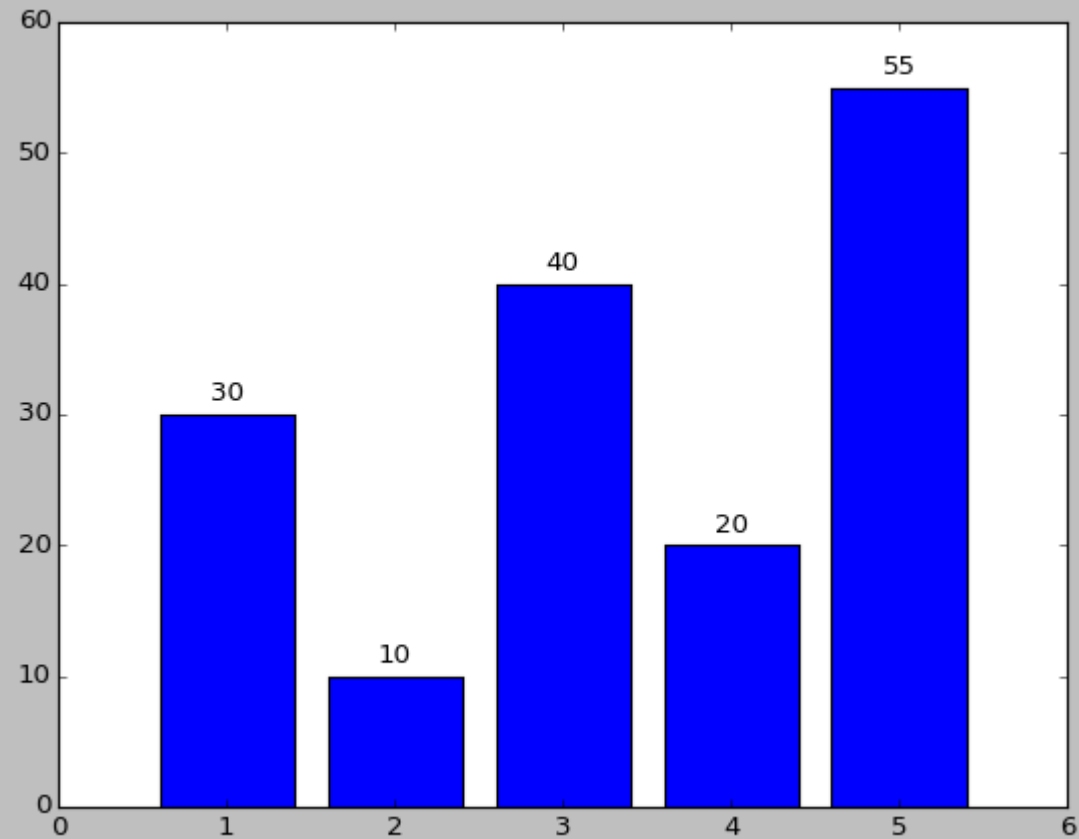
Afin d'aligner chaque valeur à sa barre respective, on utilisera la liste « position » que l'on a utilisé pour créer le diagramme.

Un alignement « centré » pour bien tout centrer à la fin !

À vous ensuite de bien « doser » les valeurs pour bien placer votre texte.

# EXAMPLE

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], align='center')  
  
for i,j in zip([1, 2, 3, 4, 5], [30, 10, 40, 20, 55]) :  
    plt.text(i, j+1, str(j), ha='center')  
  
plt.show()
```



# DIAGRAMME EN BARRES HORIZONTAL

**`plt.barh(position, valeurs, ...)`**

Les x deviennent les y et les y deviennent les x, attention ! Les arguments à placer dans les « ... » sont les mêmes que pour un diagrammes en barres vertical, ainsi que les courbes.



# RAPIDE EXEMPLE

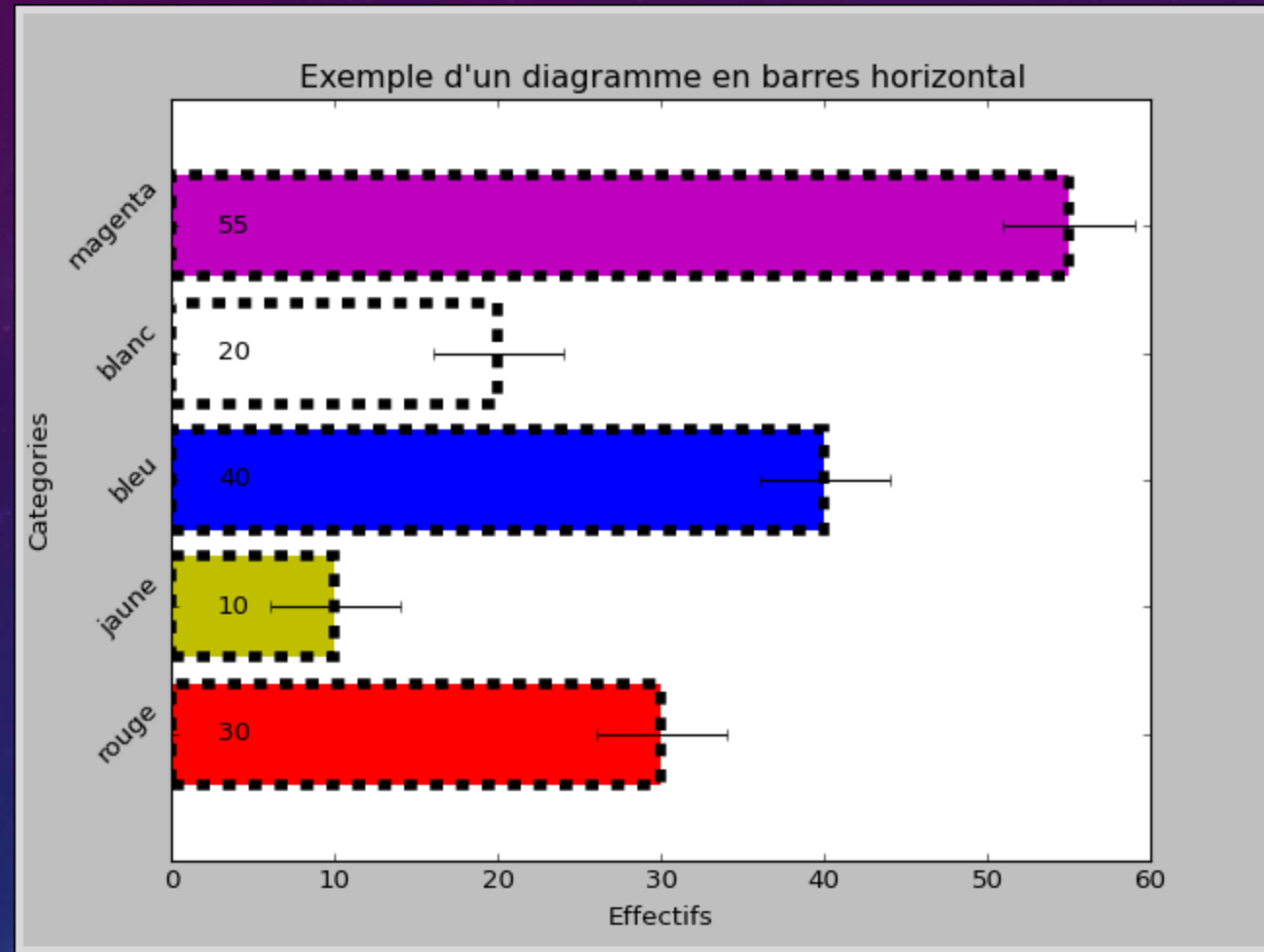
```
import matplotlib.pyplot as plt

plt.barh([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], align='center', color=["red", "y", "blue", "white",
"m"], xerr=4, ecolor="k", linewidth=5, edgecolor="k", linestyle="dashed")

for i,j in zip([1, 2, 3, 4, 5], [30, 10, 40, 20, 55]) :
    plt.text(3, i, str(j), va="center")

plt.title("Exemple d'un diagramme en barres horizontal")
plt.xlabel("Effectifs")
plt.ylabel("Categories")
plt.yticks([1, 2, 3, 4, 5], ["rouge", "jaune", "bleu", "blanc", "magenta"], rotation=45)

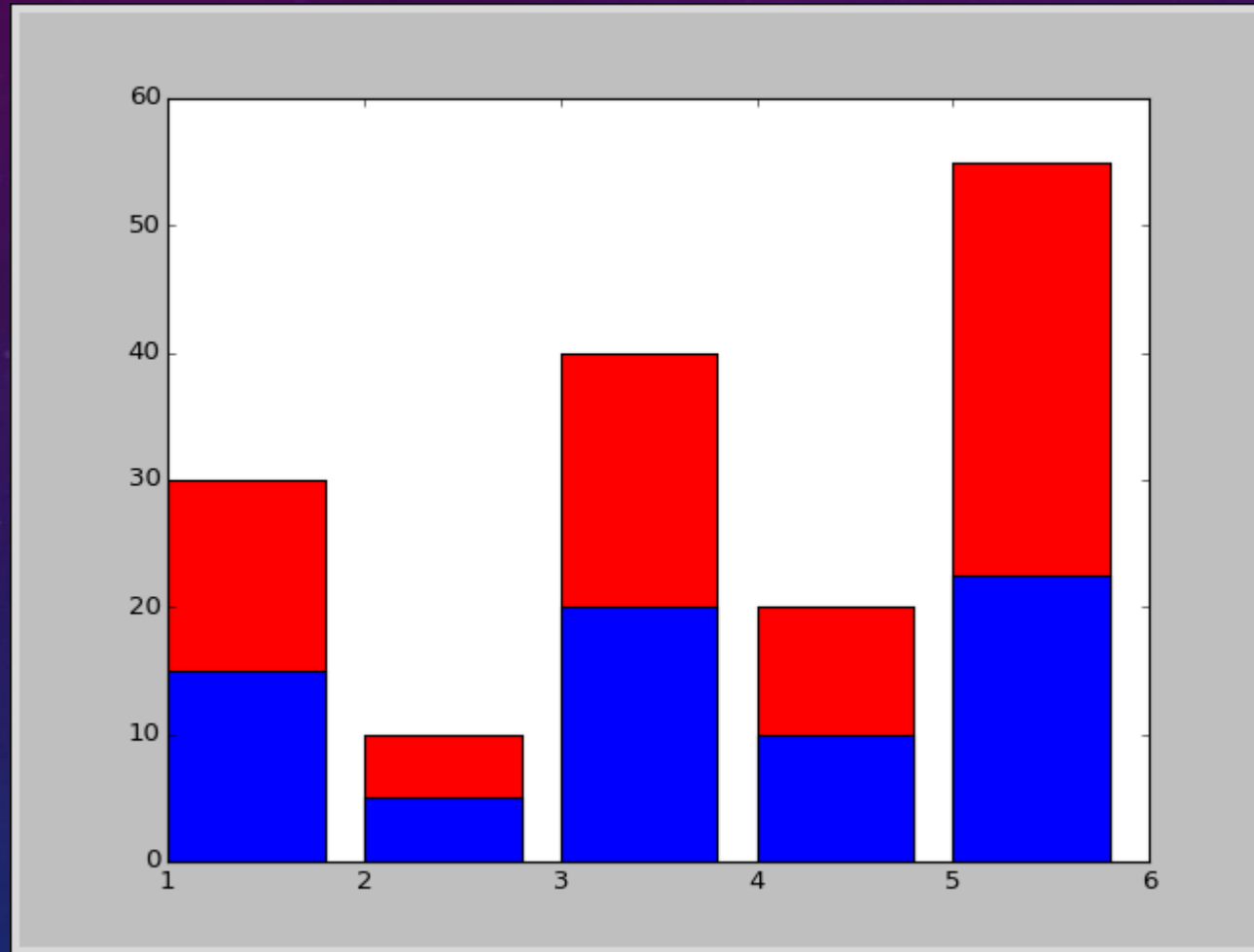
plt.show()
```



# DONNÉES MULTIPLES

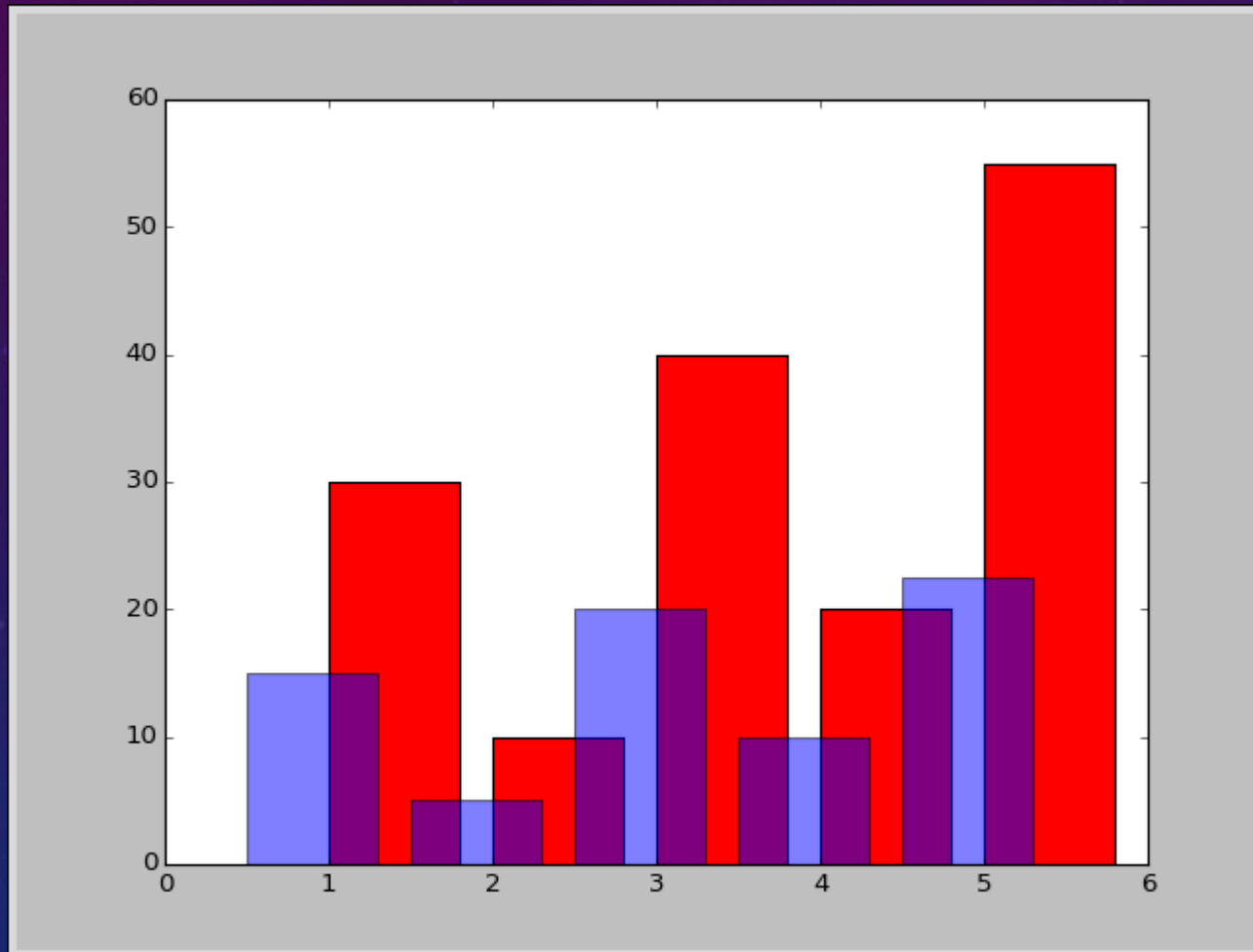
Vous pouvez afficher plusieurs séries de données sur un même graphe comme vous afficheriez plusieurs courbes sur un graphe. Dans le cas des barres, jouez sur la position des barres selon l'axe des x ou y pour ne pas les superposer (et éviter d'en cacher), ainsi que la transparence.

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], color='r')  
plt.bar([1, 2, 3, 4, 5], [15, 5, 20, 10, 22.5])  
plt.show()
```



Données superposées,  
peut mieux faire ...

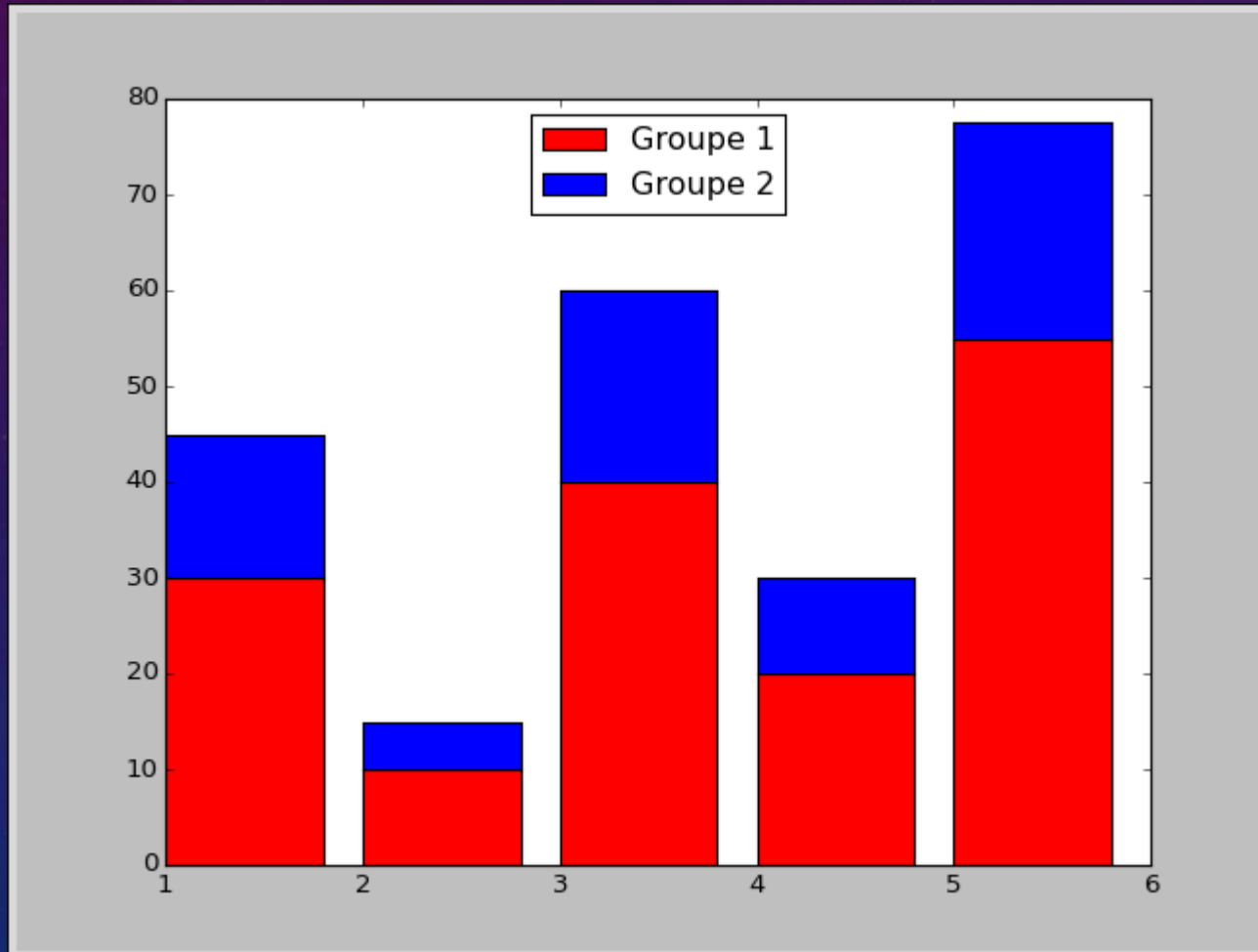
```
L = [1, 2, 3, 4, 5]
plt.bar(L, [30, 10, 40, 20, 55], color='r')
plt.bar([i-0.5 for i in L], [15, 5, 20, 10, 22.5], alpha=0.5)
plt.show()
```



Hmmmm ...

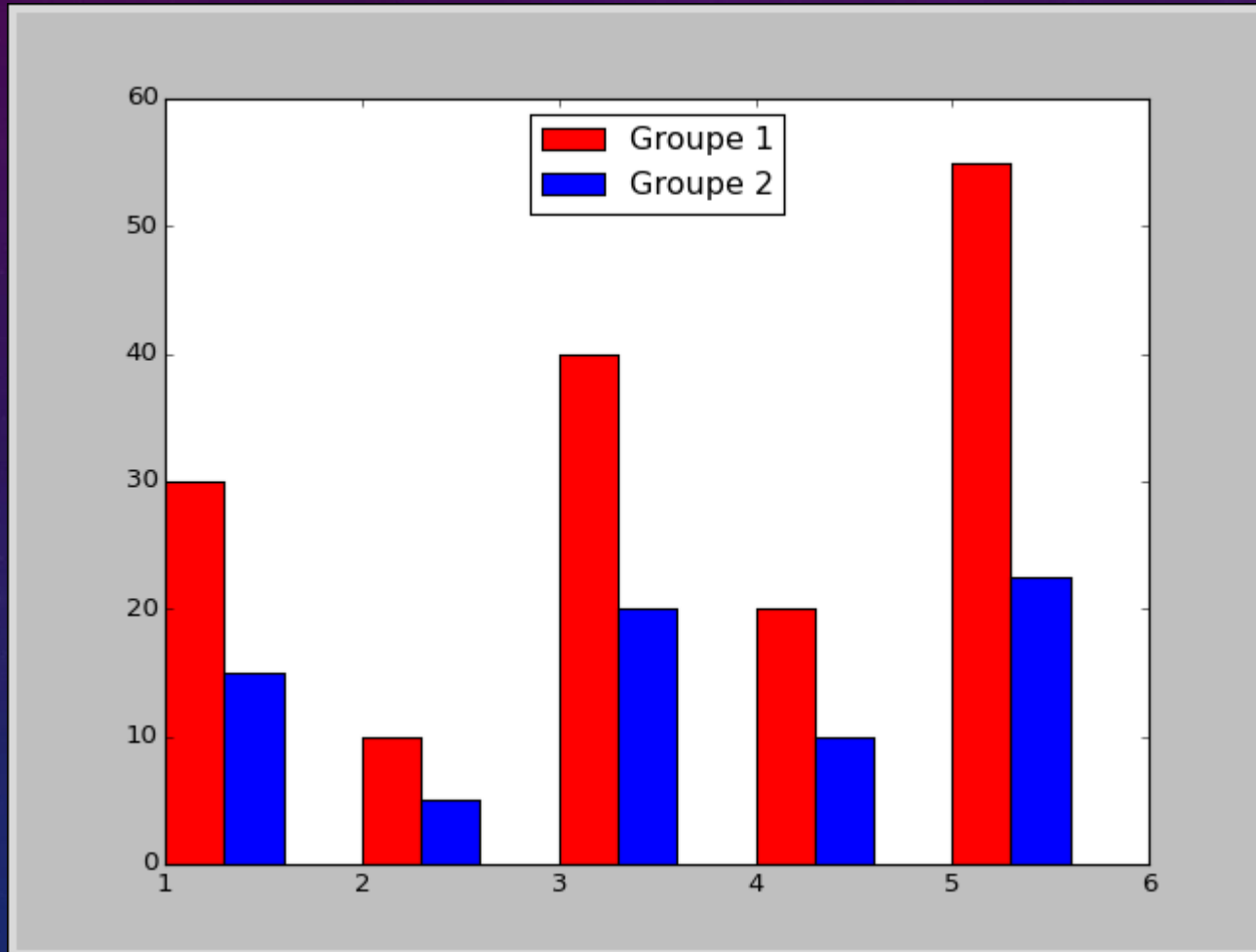


```
L = [1, 2, 3, 4, 5]
plt.bar(L, [30, 10, 40, 20, 55], color='r', label="Groupe 1")
plt.bar(L, [15, 5, 20, 10, 22.5], bottom=[30, 10, 40, 20, 55], label="Groupe 2")
plt.legend(loc=9)
plt.show()
```



Données empilées,  
mouais ...

```
plt.bar([1, 2, 3, 4, 5], [30, 10, 40, 20, 55], color='r', width=0.3, label="Groupe 1")  
plt.bar([1.30, 2.30, 3.30, 4.30, 5.30], [15, 5, 20, 10, 22.5], width=0.3, label="Groupe 2")  
plt.legend(loc=9)  
plt.show()
```



C'est mieux !

# Histogrammes

The background is a gradient of dark blue to purple, speckled with white dots resembling a starry sky. On the right side, there are faint, light-colored geometric patterns: a large circular scale with degree markings (90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210) and concentric circles, and a smaller circular diagram with dashed lines and arrows indicating a cycle or process.

# DIFFÉRENCE AVEC DIAGRAMME EN BARRES

C'est également un diagramme en barres ! Mais les données sont différentes. En effet au lieu de spécifier directement la hauteur des barres, vous aurez à la place une liste valeurs. L'histogramme va regarder chaque valeur et les placer dans une catégorie. Si la catégorie existe déjà, le nombre d'éléments qui composent cette catégorie est incrémenté de 1. Sinon elle est créée, et mise à 1 (c'est comme l'existence ou non d'une clé dans un dictionnaire, voir cours sur les dictionnaires).

La hauteur des barres n'est donc pas connue en avance, mais seulement lorsque la liste de valeurs a été entièrement lue.



# QUAND UTILISER UN HISTOGRAMME ?

L'histogramme est utilisé pour visualiser la distribution d'effectifs. C'est un comptage. Le mieux serait de prendre directement un exemple : un lancé de dé. Le dé est lancé un certain nombre de fois et les nombres qui sortent sont gardés dans une liste par exemple, puis nous pourrions visualiser combien de 1 sont sortis, combien de 2, etc.

Pour créer un histogramme, la syntaxe est la suivante :

**`plt.hist(donnees, bins, ...)`**

- « `donnees` » est la liste qui contiendra tous les résultats du dé pour cet exemple. Des arguments peuvent venir compléter le graphe (légende, couleur, etc.), qui sont pratiquement les mêmes que pour les courbes et diagrammes en barres.
- « `bins` » est le nombre de barres que vous voulez afficher. Il serait logique de prendre  $x = 6$  (pour le dé), mais vous pouvez mettre 5 ou moins (discrétisation). Vous pouvez également ne pas spécifier `bins`, dans ce cas, le nombre de barres sera déterminé automatiquement selon les données (mais vaut mieux le préciser). Il est également possible de passer une liste de valeurs, avec autant de valeurs qu'il y a de barres afin de déterminer leur « position » sur le graphe (« » parce que c'est compliqué).

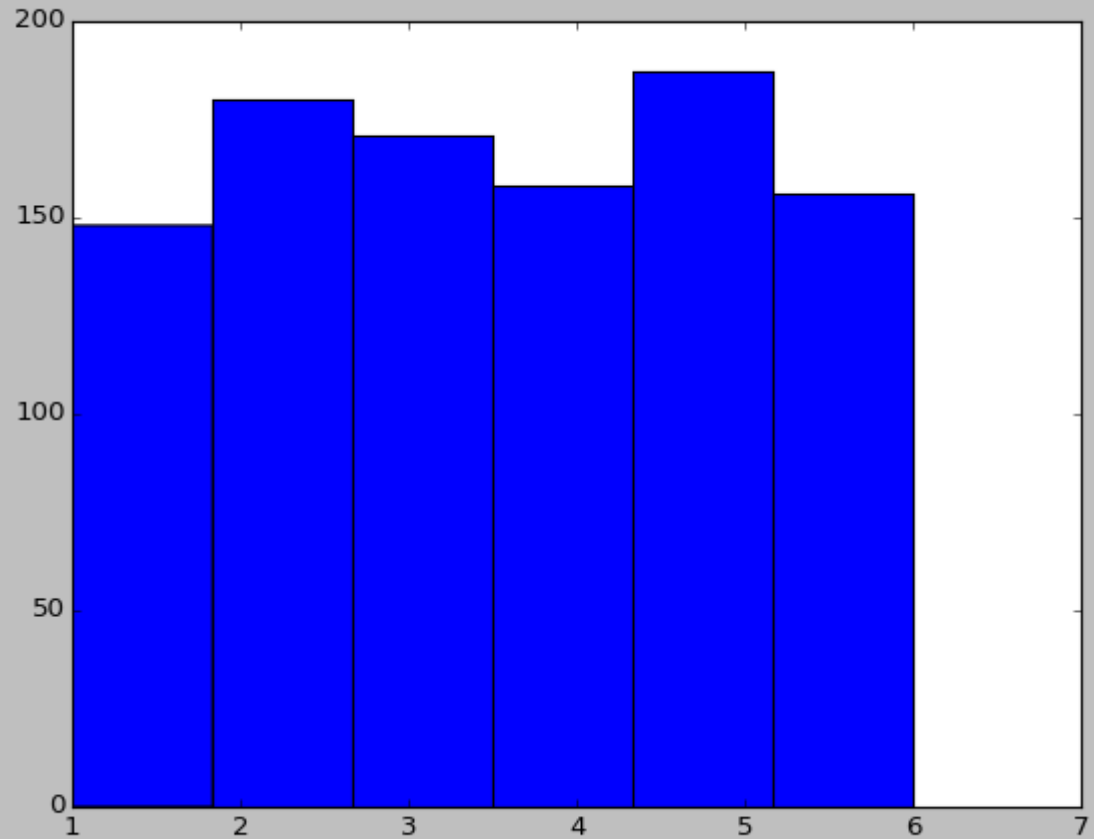


# UN PREMIER EXEMPLE

```
import matplotlib.pyplot as plt
import random

D = [random.randint(1, 6) for i in range(1000)]
plt.hist(D, 6)

plt.show()
```



Comme vous pouvez le remarquer, ce n'est pas très bien présenté. Pour cela, vous pouvez modifier l'affichage à l'aide d'arguments et que vous insérer à la suite de vos données dans **plt.hist()**.

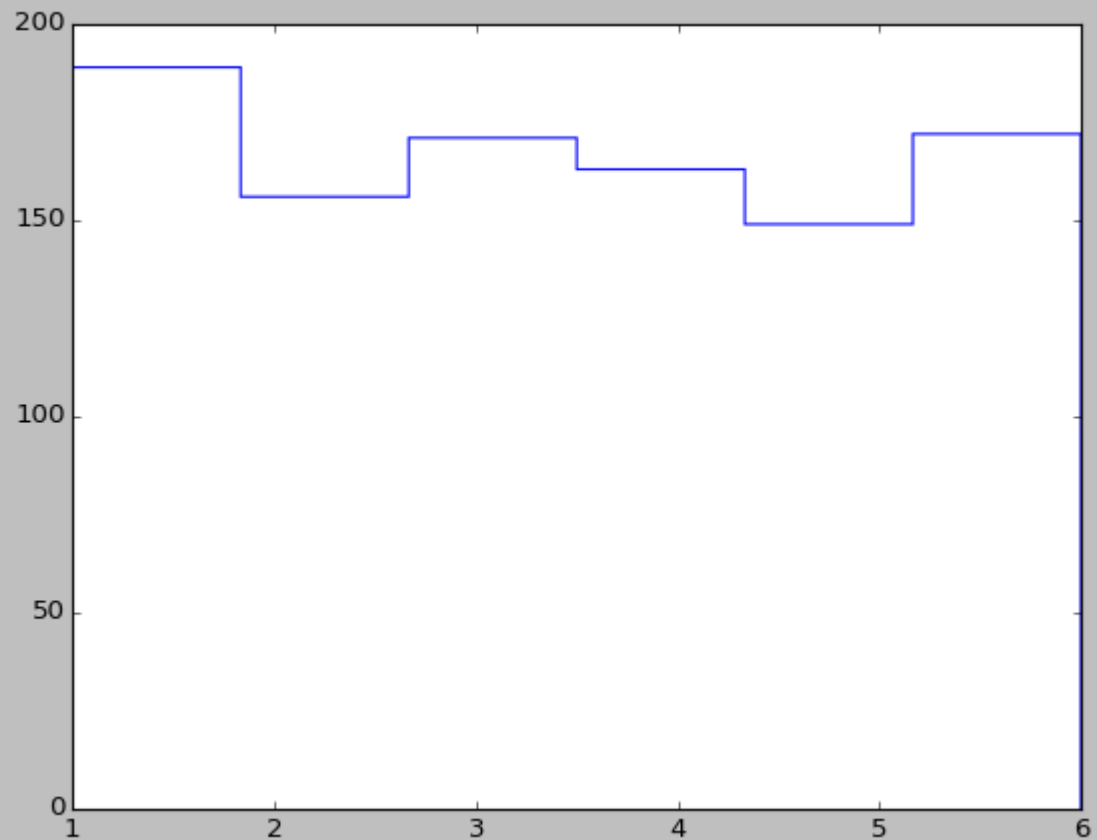
# TYPES D'HISTOGRAMME

Vous pouvez modifier l'apparence des barres le précisant avec **histtype** = « **string** ». Les types possibles sont :

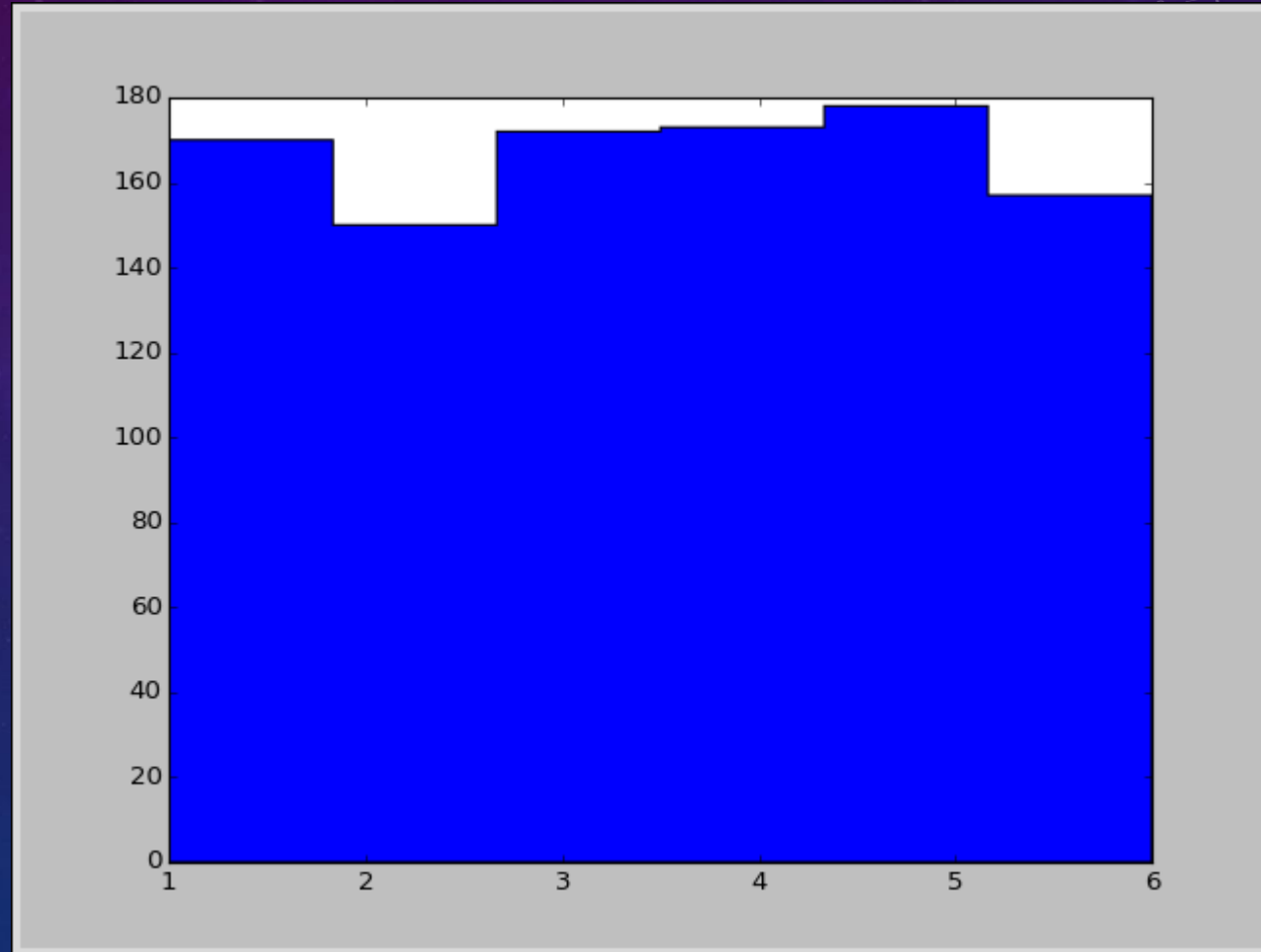
- bar (apparence par défaut)
- barstacked (dans le cas de données multiples, les barres des nouvelles données sont placés au dessus des anciennes données)
- step (une ligne faisant le contour des barres est tracé (ensemble des barres, et non pas individuels), avec un font blanc)
- stepfilled (même chose que step, mais le font est rempli)

# EXAMPLES

```
D = [random.randint(1, 6) for i in range(1000)]  
plt.hist(D, 6, histtype='step')  
plt.show()
```



```
D = [random.randint(1, 6) for i in range(1000)]  
plt.hist(D, 6, histtype='stepfilled')  
plt.show()
```





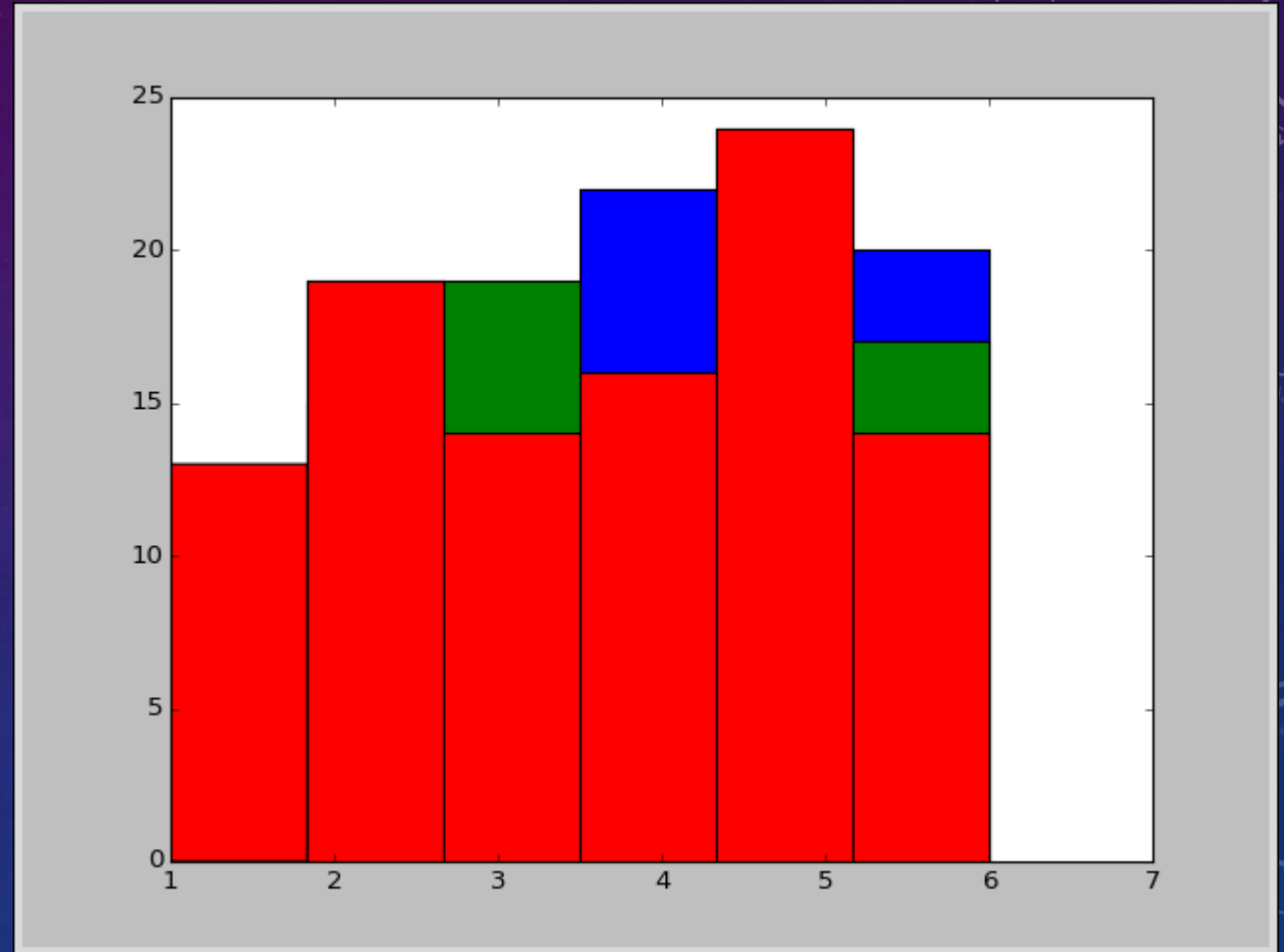
# DONNÉES MULTIPLES

Reprenons le lancé de dé. Si vous faites 3 séries de 100 lancés de dé, vous pouvez afficher vos 3 séries sur le même graphe.

Soit vous répétez 3 fois de suite **plt.hist()** pour chaque série, soit vous regroupez vos 3 listes en une liste (une liste de 3 listes, et non pas une liste de valeurs), et vous la passer dans un seul **plt.hist()**, ce qui est beaucoup plus lisible puisque les dernières barres affichées peuvent en cacher. Voyez les exemples dans les diapos suivantes.

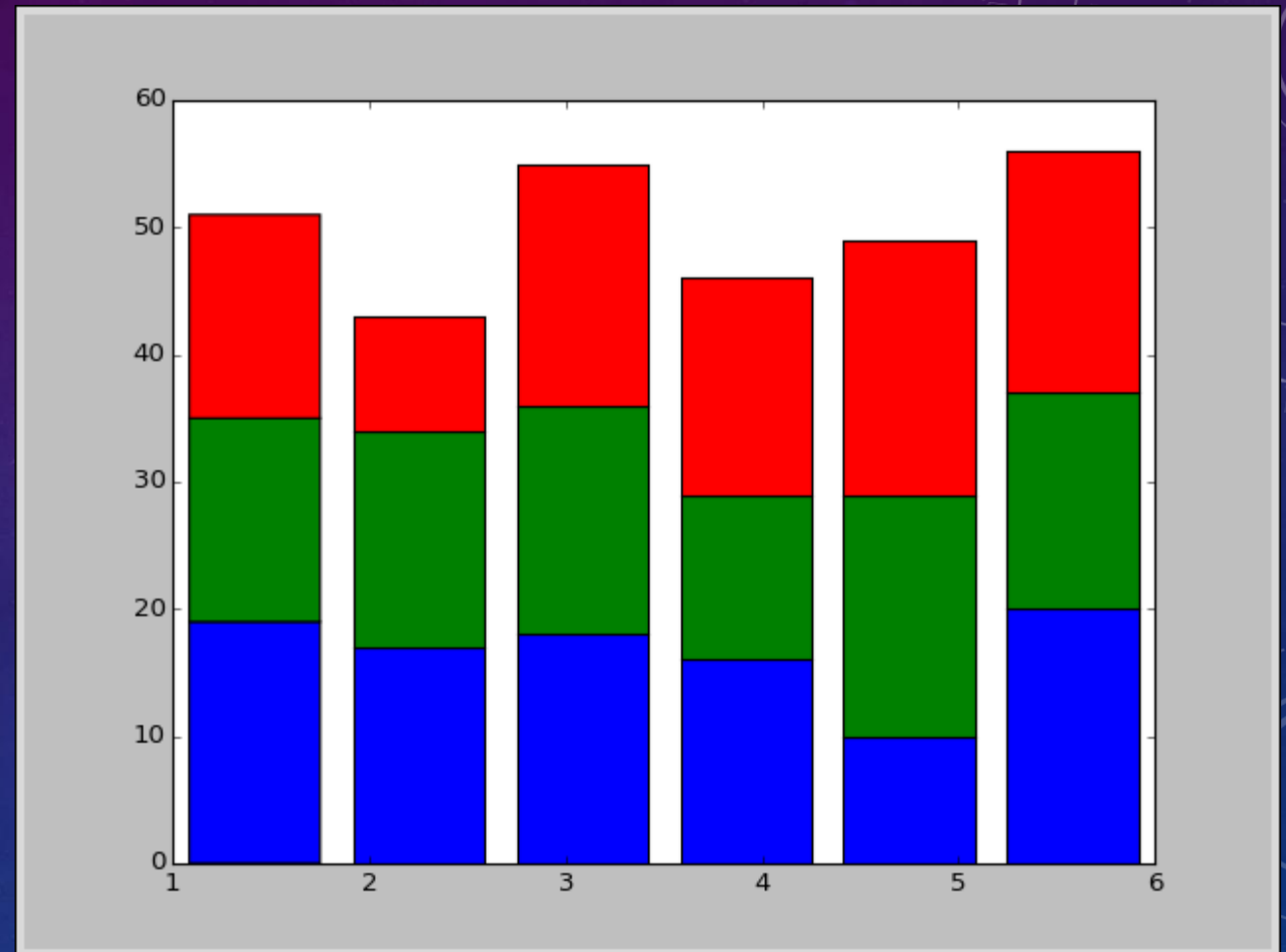
```
D = [random.randint(1, 6) for i in range(100)]  
D2 = [random.randint(1, 6) for i in range(100)]  
D3 = [random.randint(1, 6) for i in range(100)]  
  
plt.hist(D, 6)  
plt.hist(D2, 6)  
plt.hist(D3, 6)  
  
plt.show()
```

Outch !!!



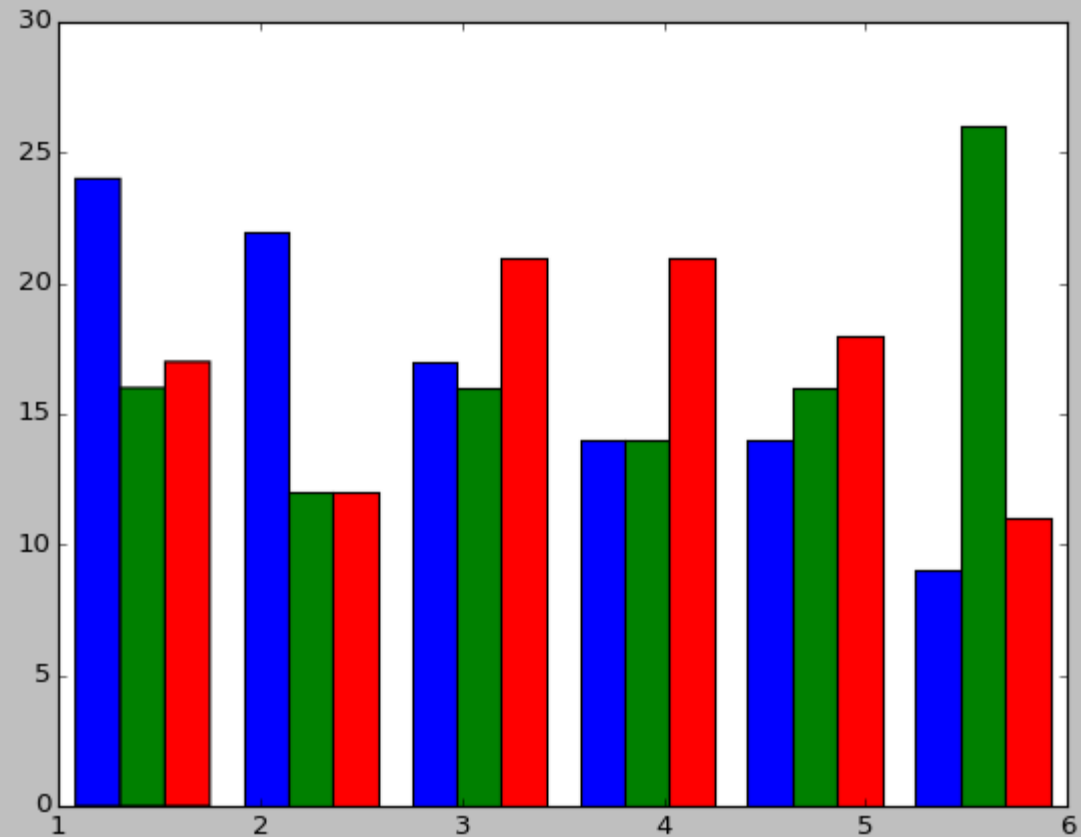
```
D = [random.randint(1, 6) for i in range(100)]  
D2 = [random.randint(1, 6) for i in range(100)]  
D3 = [random.randint(1, 6) for i in range(100)]  
  
plt.hist([D, D2, D3], 6, histtype='barstacked')  
plt.show()
```

Outch un peu moins ...  
(empilement si vous  
n'avez pas compris)



```
D = [random.randint(1, 6) for i in range(100)]  
D2 = [random.randint(1, 6) for i in range(100)]  
D3 = [random.randint(1, 6) for i in range(100)]  
  
plt.hist([D, D2, D3], 6)  
  
plt.show()
```

Mieux !

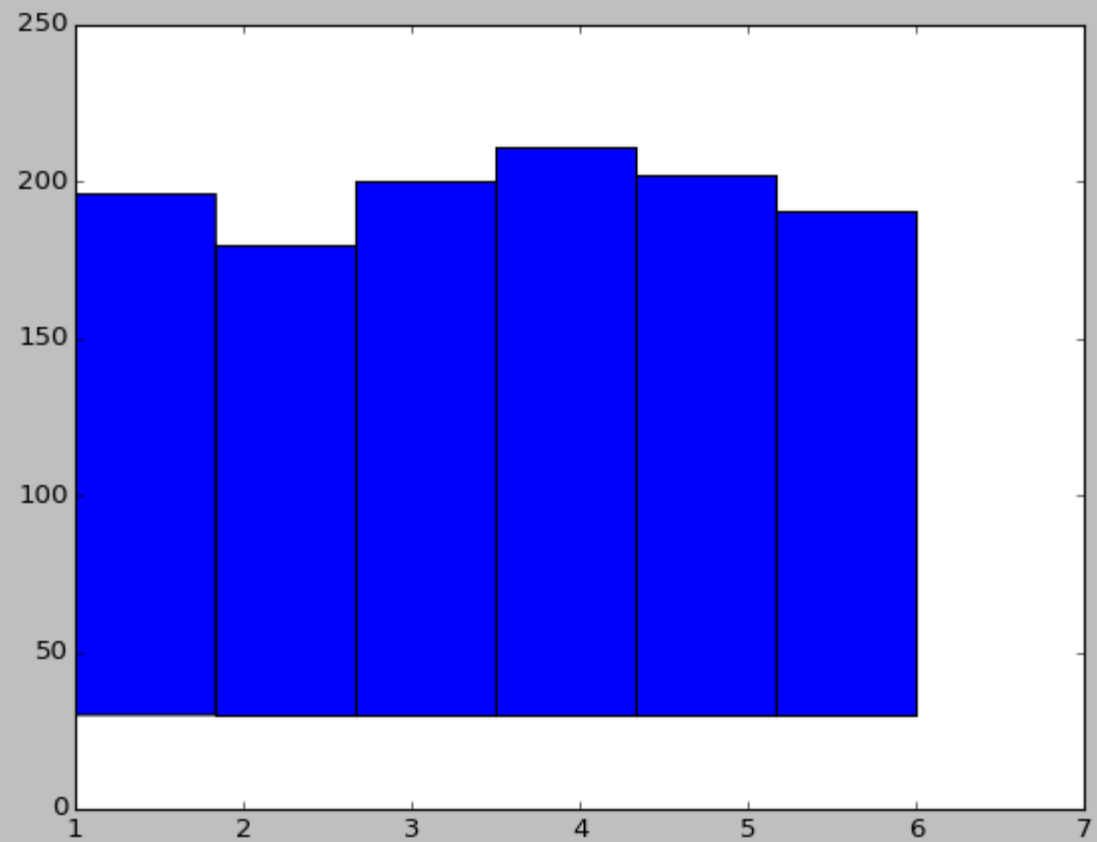


# ALIGNEMENT PAR RAPPORT À L'AXE DES Y

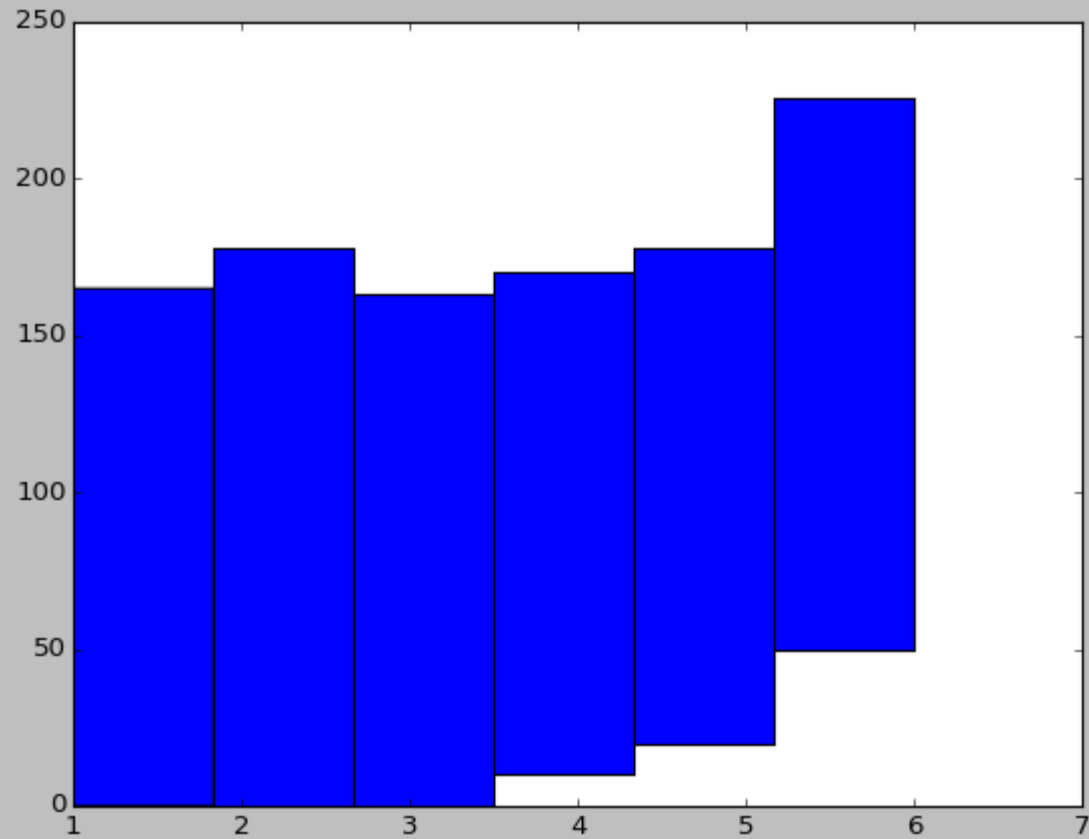
Même chose qu'avec les diagrammes en barres avec  
**bottom = int/float** (voir diapo 99).



```
D = [random.randint(1, 6) for i in range(1000)]  
plt.hist(D, 6, bottom=30)  
plt.show()
```



```
D = [random.randint(1, 6) for i in range(1000)]  
plt.hist(D, 6, bottom=[0, 0, 0, 10, 20, 50])  
plt.show()
```



# LÉGENDE

Voir la fameuse diapo

117

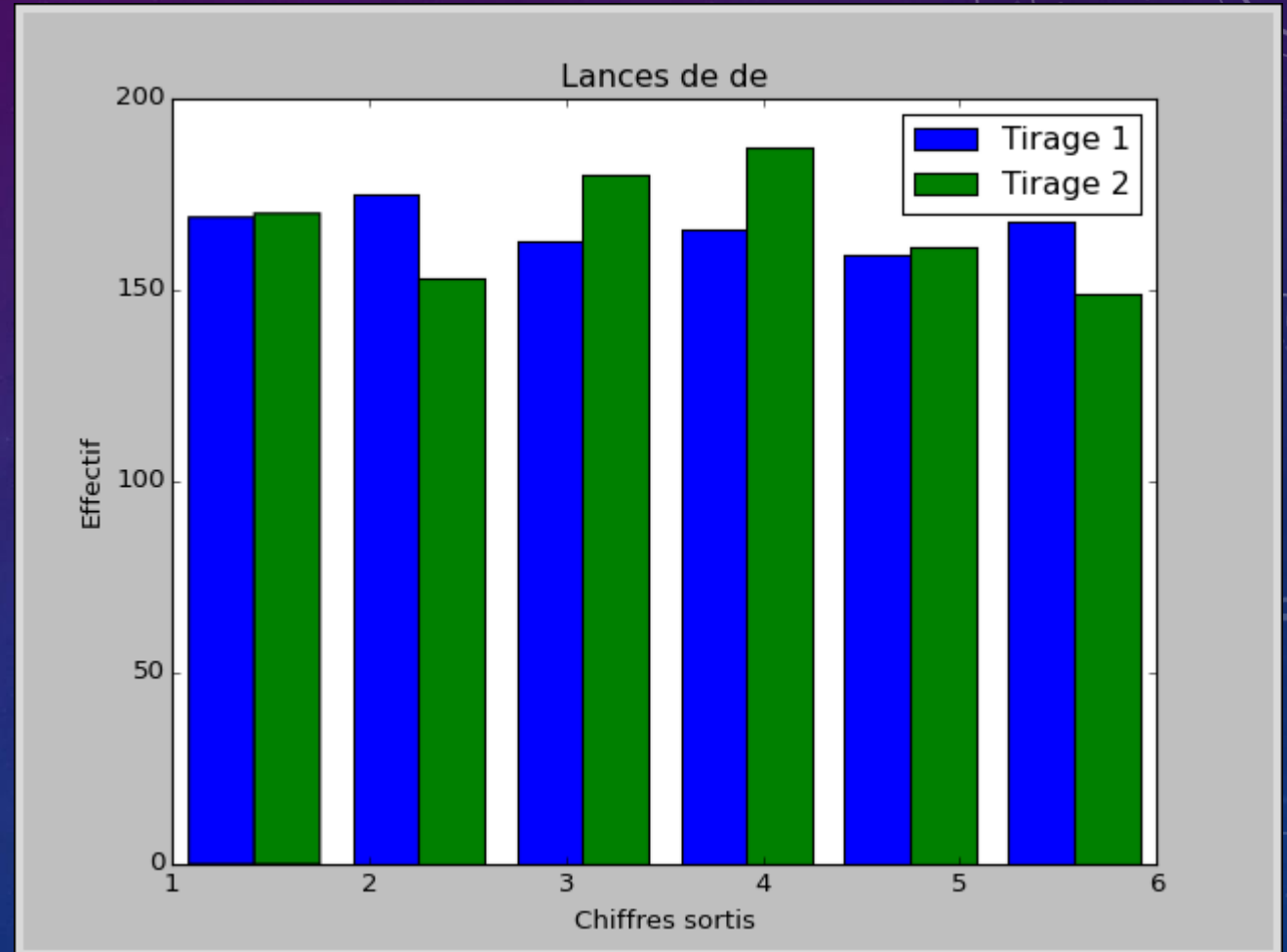
# EXEMPLE

```
D = [random.randint(1, 6) for i in range(1000)]
D2 = [random.randint(1, 6) for i in range(1000)]

plt.hist([D, D2], 6, label=["Tirage 1", "Tirage 2"])

plt.legend()
plt.title("Lances de de")
plt.xlabel("Chiffres sortis")
plt.ylabel("Effectif")

plt.show()
```



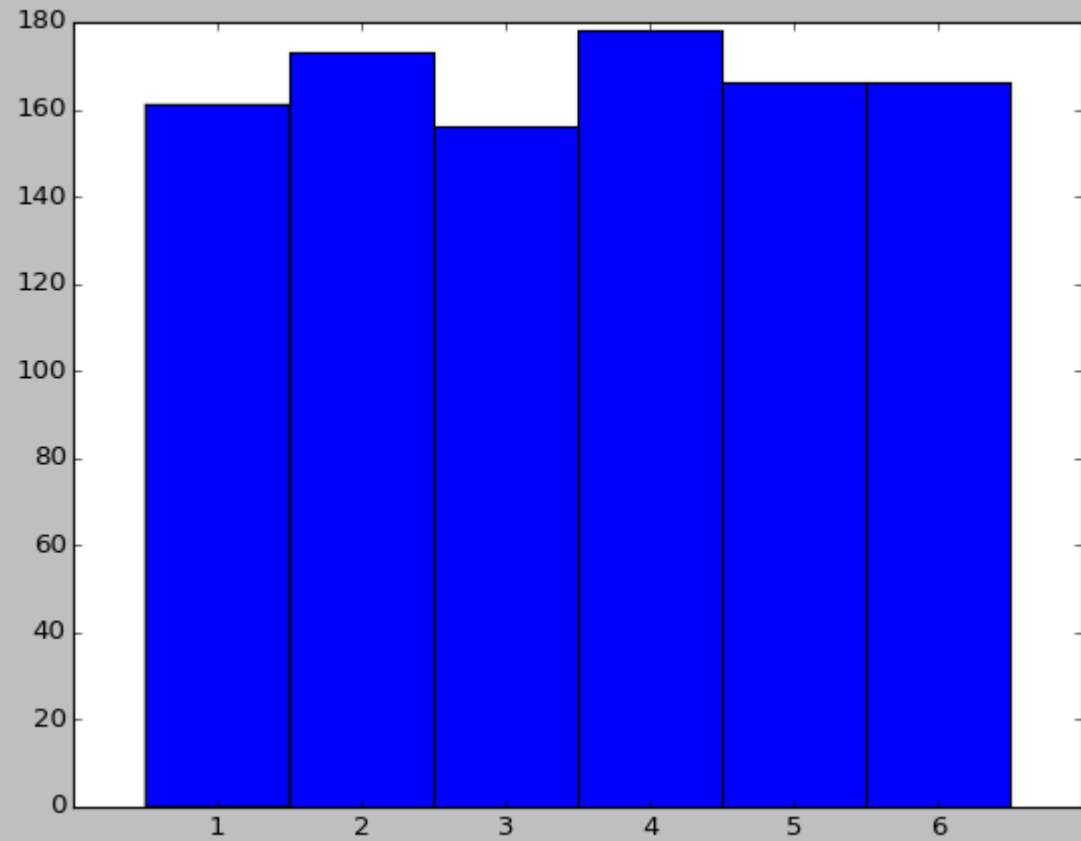
# ALIGNEMENT DE LA GRADUATION DE L'AXE DES X AVEC LES BARRES

Vous avez pu remarquer que la graduation des x avec les barres n'est pas parfait. En effet, pour les histogrammes, cela n'est pas vraiment important, et c'est assez compliqué. Il est toutefois possible de le faire avec quelques manipulations.



# UNE FAÇON DE FAIRE

```
D = [random.randint(1, 6) for i in range(1000)]  
plt.hist(D, bins=[0.5+i for i in range(7)])  
plt.xticks(range(1, 7))  
plt.xlim(0, 7)  
plt.show()
```



# HAUTEUR DES BARRES

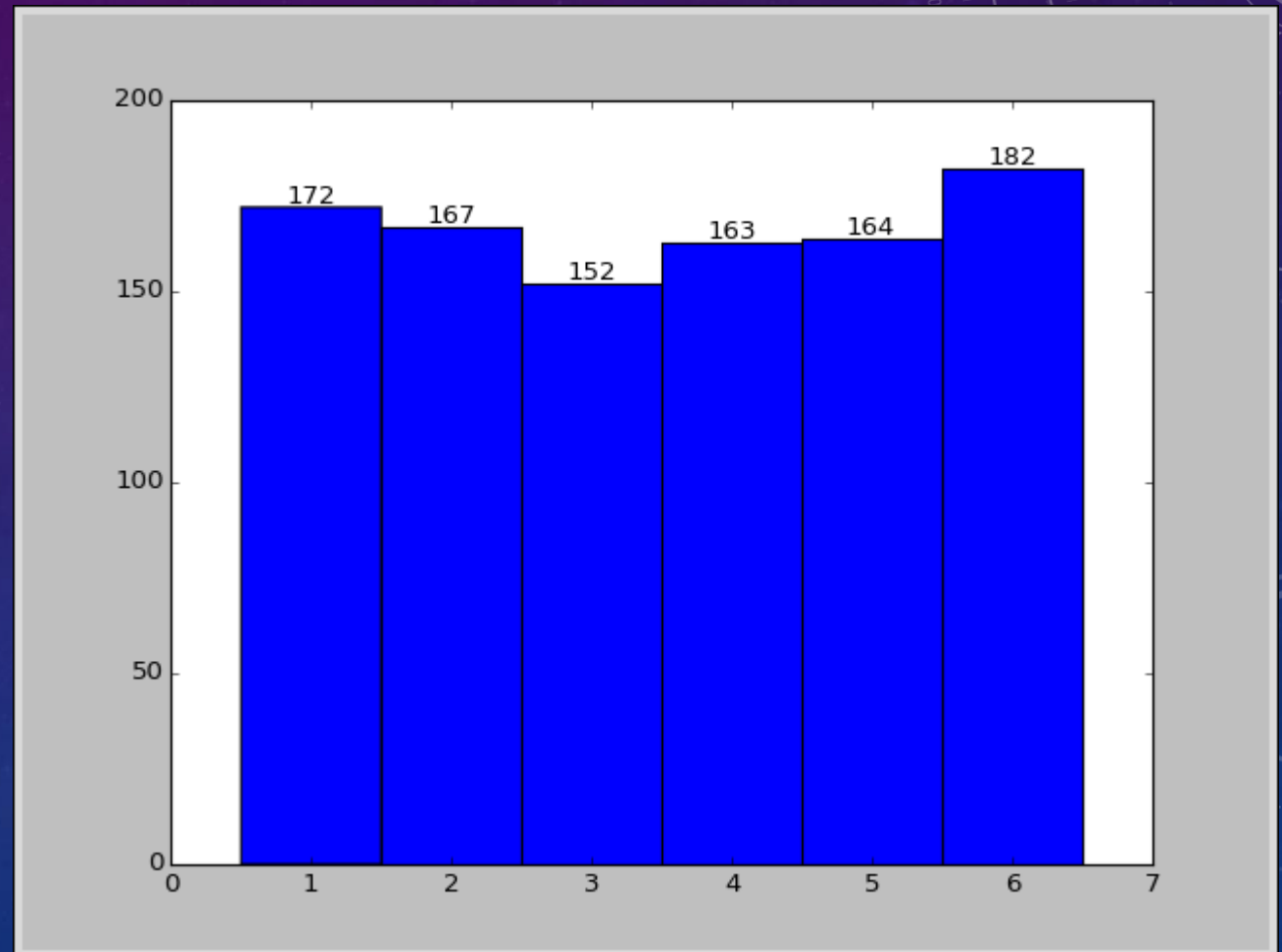
Si vous vous souvenez de ce qui a été raconté dans la diapo 50, 101 et 102 à propos de la hauteur des barres, pour les histogrammes, il est possible de faire manipulation avec **plt.text()**, mais cela ne se fait pas. En effet, les données sont une liste de résultats, de chiffres, ... La hauteur et le nombre de barres ne sont connus que lorsque l'on a fini de parcourir les données.

Il faudrait faire une fonction de comptage, ou de le faire manuellement. Mais sachez que cela ne se fait pas ! Nada ! Point barre ! Fin de l'histoire ! Tchao !

# UNE PETITE ILLUSTRATION POUR CEUX QUI INSISTENT ...

```
p = [random.randint(1, 6) for i in range(1000)]  
plt.hist(p, [0.5+i for i in range(7)])  
L = super_fonction_de_comptage(p)  
  
for a,b in zip(L, range(1, 7)):  
    plt.text(b, a+1, str(a), ha='center')  
  
plt.show()
```

À vous d'écrire la super fonction de comptage, vous avez tous les outils en main ! (votre clavier suffit en fait)

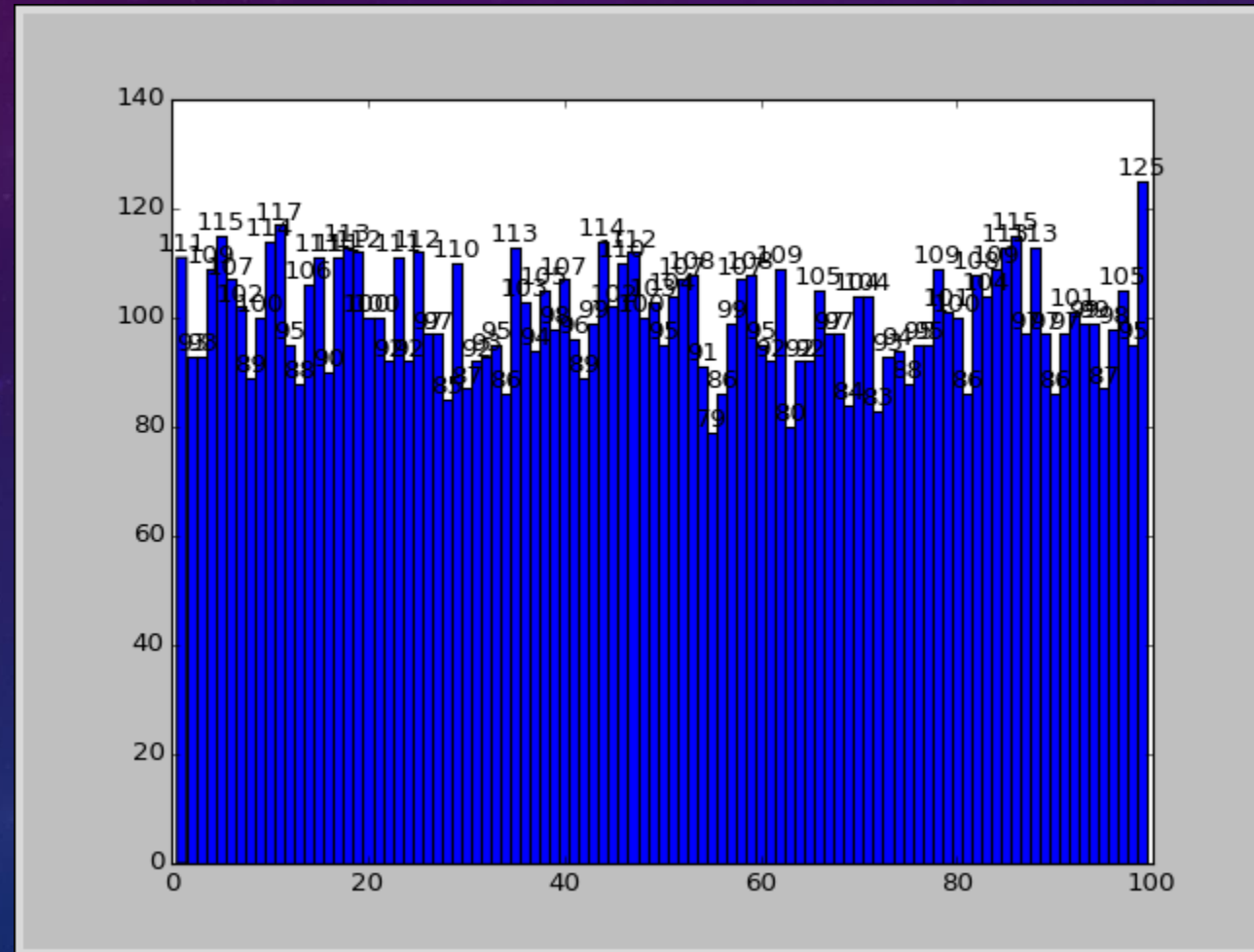


Pour ceux et celles qui ont la flemme :

```
def super_fonction_de_comptage(liste):  
    dico = {}  
    for i in liste :  
        if dico.has_key(str(i)) :  
            dico[str(i)] += 1  
        else:  
            dico[str(i)] = 1  
  
    D = dico.items()  
    for j in range(len(D)):  
        D[j] = [int(D[j][0]), D[j]]  
  
    D = sorted(D)  
    for k in range(len(D)):  
        D[k] = D[k][1][1]  
  
    return D
```

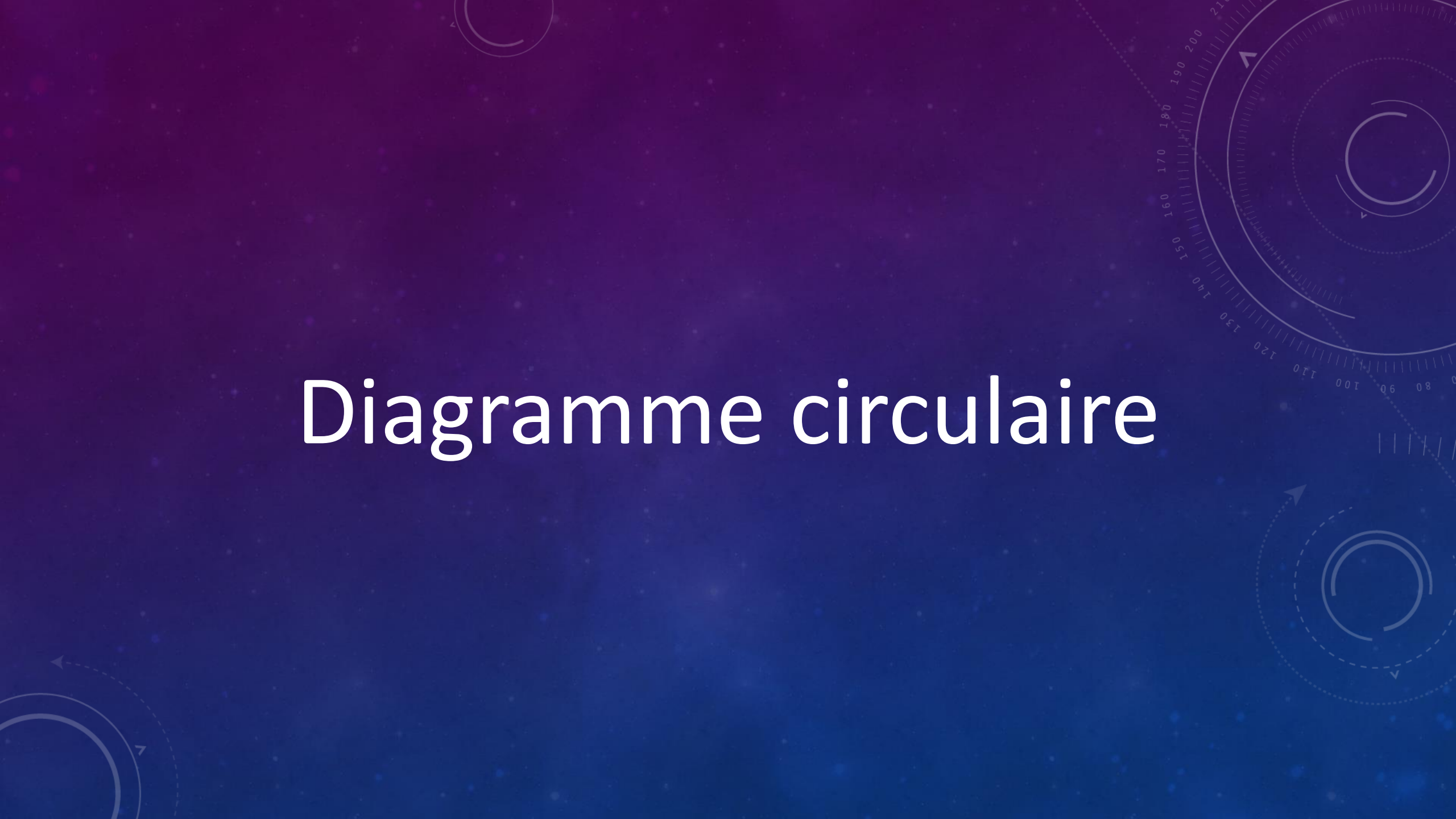
Il faudra malheureusement l'écrire si vous voulez l'utilisez ! Je vous ai déjà enlevé la dure tâche qui est de réfléchir ! Si vous ne savez pas créer de fonction, je vous invite à relire votre cours sur les définitions de fonctions.

Imaginez si vous voulez faire de même avec un histogramme de 100 barres par exemple. Ce serait totalement illisible ! C'est pour cela que cela ne se fait pas pour les histogrammes.





# Diagramme circulaire



Le diagramme circulaire ! Ou plus communément appelé ... ~~la pizza~~ – le camembert !

Pourquoi ? Parce que cela ressemble à un camembert, sans l'odeur.  
Ça aurait pu s'appeler une pizza, ou une tarte, mais ils (qui donc ?) ont choisi le fameux camembert. Contemplez cette belle croûte blanchâtre habillant une texture onctueuse, crémeuse et ... BREF !

Bien qu'appelé camembert, la fonction permettant de tracer un diagramme circulaire s'appelle « **pie** », soit « tarte » en anglais ...  
Donc en anglais on dirait : « Draw me a pie ! » Soit : « Dessine moi une tarte ! »  
Ça porte à confusion tout cela .....



En soit, ce qui nous intéresse c'est comment obtenir le diagramme.

Toujours avec notre « **import matplotlib.pyplot as plt** », la syntaxe est la suivante :

**plt.pie(donnees, ...)**

Le premier paramètre que prend la fonction **pie** est une liste de valeurs qui dont les données à représenter. Il faut forcément que ce soit une liste, vous ne pouvez pas passer une valeur unique en paramètre. Si vous souhaitez le faire, il suffit de faire une liste, avec une seule valeur. Vous pouvez ensuite mettre autant de valeurs que vous voulez dans la liste.

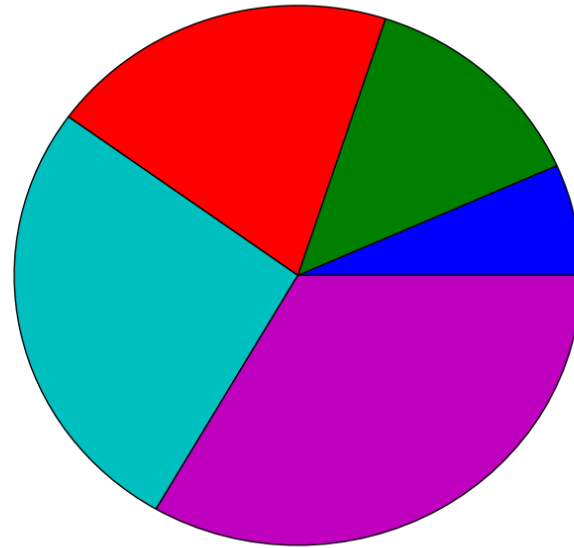
Contrairement aux courbes et diagrammes en barres, dans lesquelles l'axe des x (horizontale) tend vers l'infini, un diagramme circulaire, est un disque, de 360°. Il faut donc une limite.

Là vous vous dites, que la somme des valeurs de la liste des données doit être égale à 360, ou à 1, ou à 100, ou à une quelconque autre valeur mais non. La fonction calcule d'elle-même le pourcentage de chaque variable rapporté à la somme totale des variables, et se débrouille pour tout placer dans la tar- ... le camembert.

Magique ! N'est-ce pas ???????????

# EXAMPLES !

```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5])  
plt.show()
```





```
1 import matplotlib.pyplot as plt
2
3 plt.pie(1)
4 plt.show()
```

Traceback (most recent call last):

File "test.py", line 3, in <module>

plt.pie(1)

File "/usr/lib/python2.7/dist-packages/matplotlib/pyplot.py", line 3137, in pie  
frame=frame, data=data)

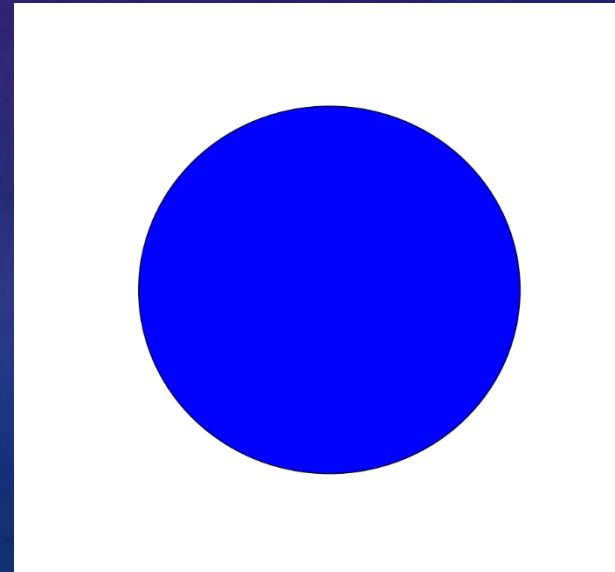
File "/usr/lib/python2.7/dist-packages/matplotlib/\_\_init\_\_.py", line 1814, in inner  
return func(ax, \*args, \*\*kwargs)

File "/usr/lib/python2.7/dist-packages/matplotlib/axes/\_axes.py", line 2555, in pie  
labels = [''] \* len(x)

TypeError: len() of unsized object

```
import matplotlib.pyplot as plt

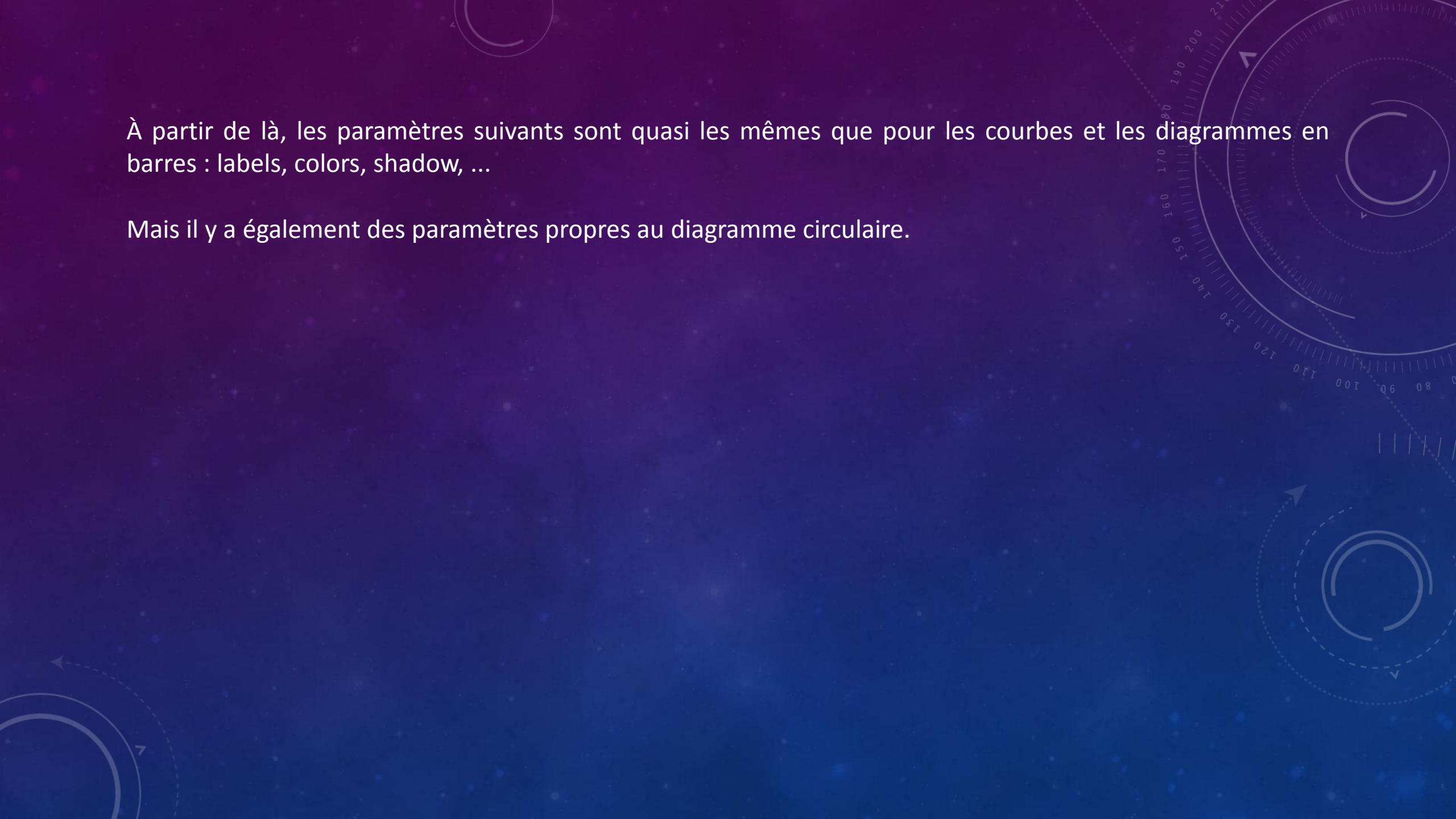
plt.pie([1])
plt.show()
```





À partir de là, les paramètres suivants sont quasi les mêmes que pour les courbes et les diagrammes en barres : labels, colors, shadow, ...

Mais il y a également des paramètres propres au diagramme circulaire.



# DÉTACHEMENT D'UNE OU PLUSIEURS PARTIES

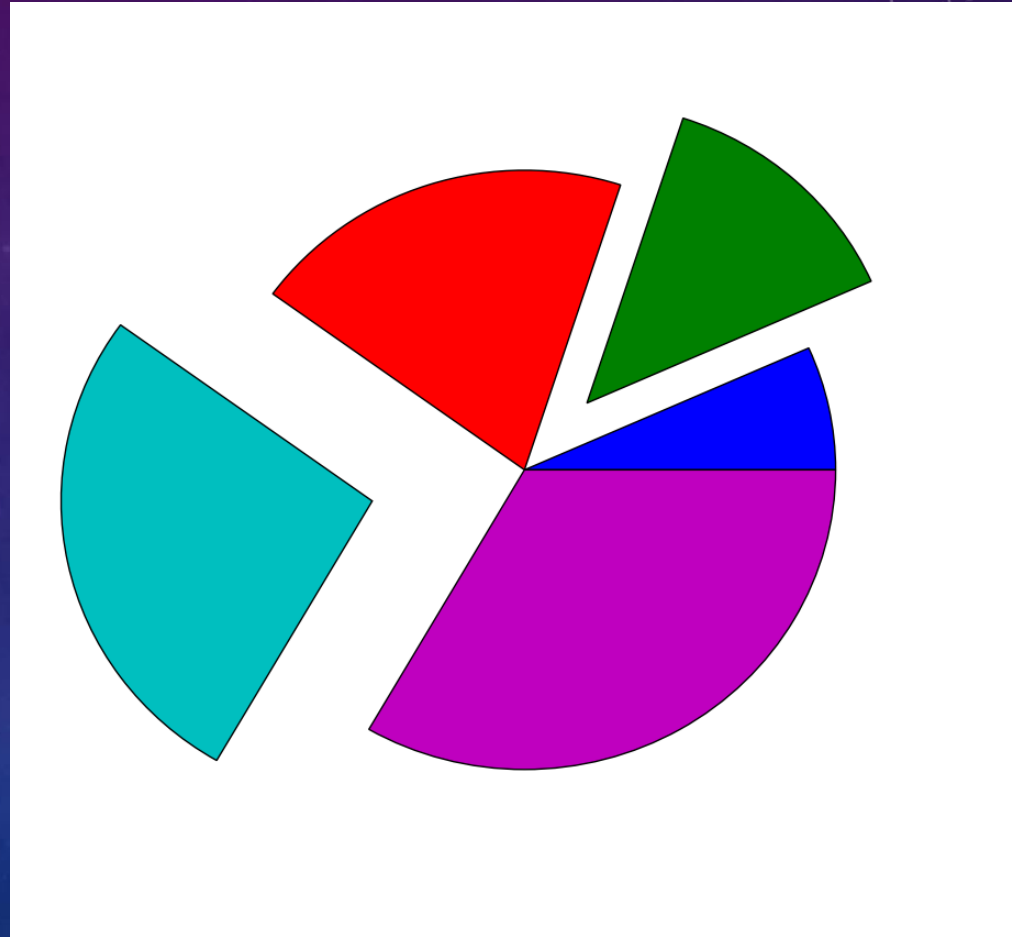
Prenons l'exemple d'une tarte. Une personne normalement constituée ne mangerait pas une tarte toute entière. Ce serait trop généreux. De même que pour un camembert. Vous allez donc en prendre une part, ou plusieurs. C'est le principe ici.

La paramètre à utiliser ici est **explode**, ou « exploser » en français. Que prend-il ? Une liste de valeurs. Cette liste doit être de même dimension que celle de données. Chaque valeur de **explode** correspondra donc au détachement de chaque valeur.

Ces valeurs peuvent être négatives, positives, grandes ou petites. Mais je vous conseille de rester dans la gamme  $[0, 1]$ . Au delà et en dessous, vous ferez du hors cadre ou du rentre dedans et cela n'a pas d'intérêt. Par défaut, toutes les valeurs sont égales à 0, mais il faut quand même les préciser qu'elles sont à 0 si vous en modifier ne serait-ce qu'une.

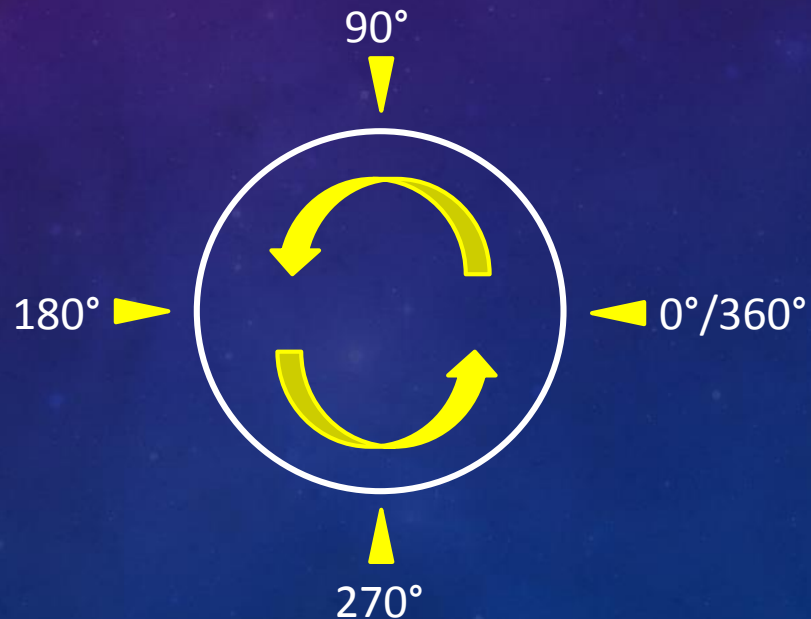
# EXAMPLE

```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5], explode=[0, 0.3, 0, 0.5, 0])  
plt.show()
```



# ANGLE DE DÉPART

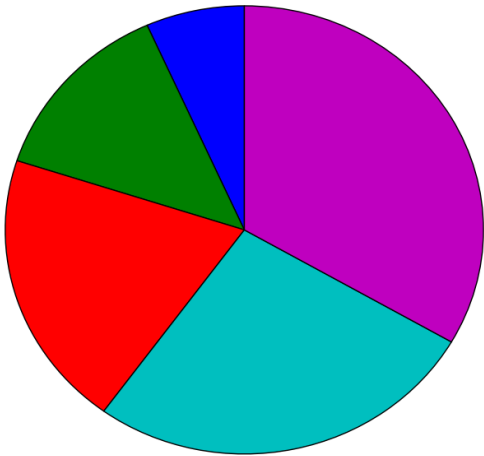
Vous pouvez spécifier l'angle de départ de vos données dans le diagramme avec le paramètre **startangle**. Il faut lui passer une valeur, valeur correspondant à l'angle duquel vous souhaitez partir. L'angle est en degré et il peut dépasser 360° auquel cas, 365° correspondra à 5°. Si vous souhaitez utiliser des radians, utilisez une fonction de conversion radians > degrés (bibliothèque **math**, fonction `math.degrees(x)`, x en radians). Si non précisé, commence à 0°.



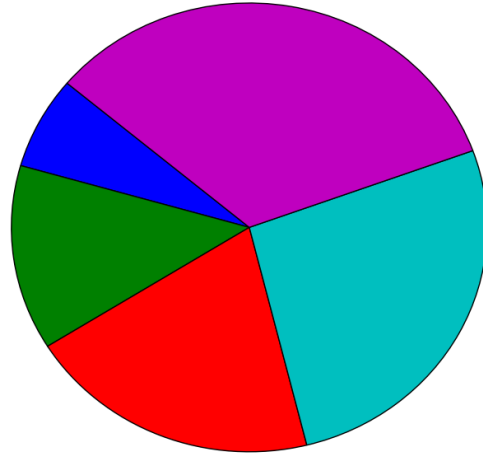


# EXAMPLES

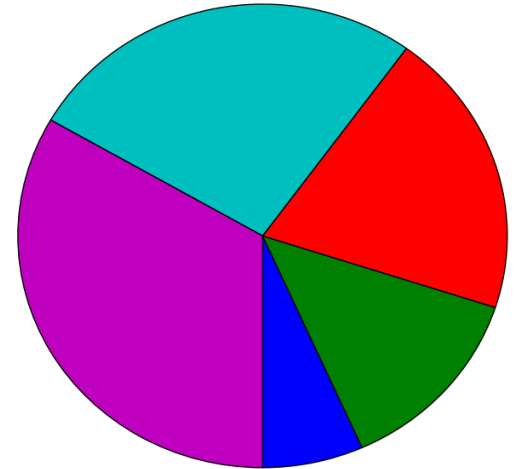
```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5], startangle=90)  
plt.show()
```



```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5], startangle=140)  
plt.show()
```



```
import matplotlib.pyplot as plt  
import math  
  
angle = math.degrees((3*math.pi)/2) # 270 degrees  
  
plt.pie([1, 2, 3, 4, 5], startangle=angle)  
plt.show()
```

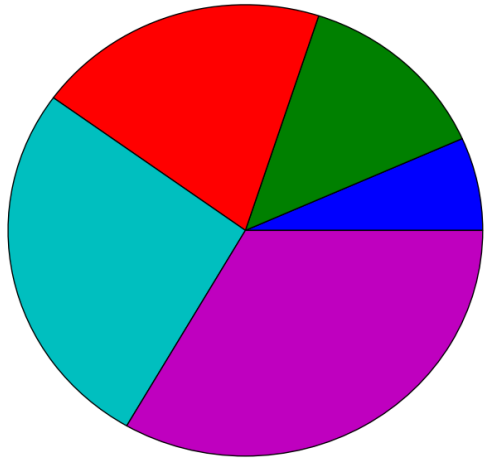




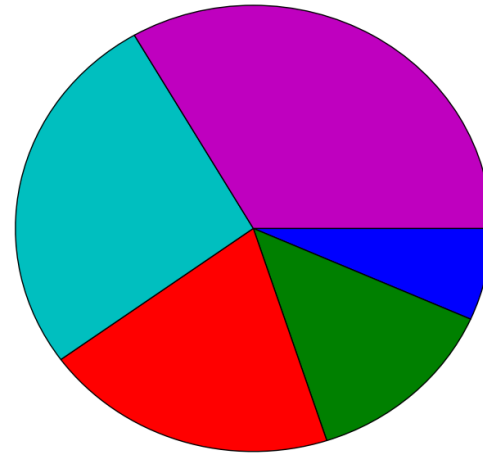
# DIRECTION

Par défaut, les données s'affichent dans le sens contraire des aiguilles d'une montre. Vous pouvez inverser le sens en le spécifiant avec **counterclock**, et en lui attribuant la valeur **False**. Par défaut **True**.

```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5])  
plt.show()
```



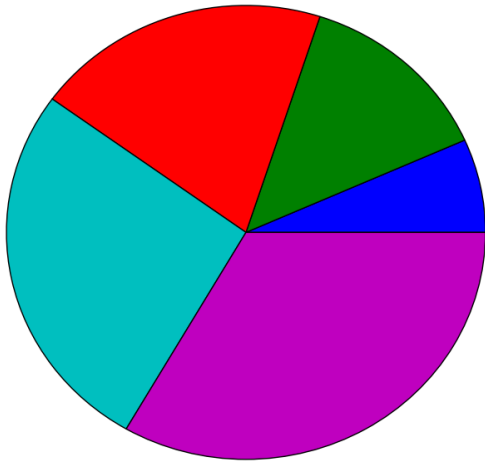
```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5], counterclock=False)  
plt.show()
```



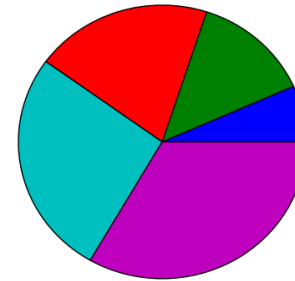
# TAILLE DU DIAGRAMME

Pour changer la taille du diagramme, il faut passer par son rayon, soit **radius**. Par défaut à 1.

```
import matplotlib.pyplot as plt  
  
plt.pie([1, 2, 3, 4, 5], radius=1)  
plt.show()
```



```
import matplotlib.pyplot as plt  
  
plt.pie([1, 2, 3, 4, 5], radius=0.6)  
plt.show()
```



# POURCENTAGE DES PARTS

Affichez le pourcentage de chaque part à l'aide de **autopct**. Il est possible de lui affecter plusieurs éléments, mais le plus simple c'est ceci '%.xf%%', avec les apostrophes. Je n'ai pas trop compris comment cela fonctionne, mais cela fonctionne. C'est une manière d'afficher une chaîne de caractères ou des nombres en python. Je n'en dirais pas plus. Mais en gros, remplacez 'x' par le nombre de décimales que vous voulez après la virgule.

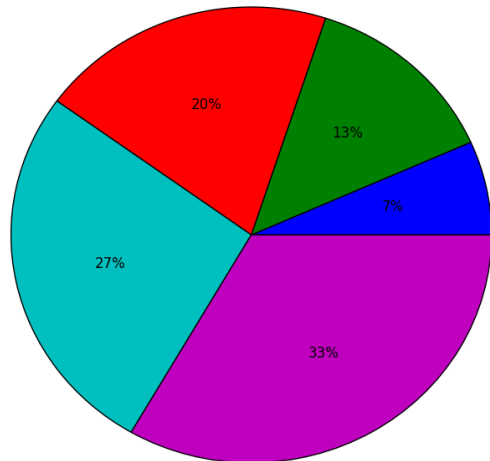
'%.0f%%' : aucun chiffre après la virgule (15%)

'%.1f%%' : un chiffre ..... (15,0%)

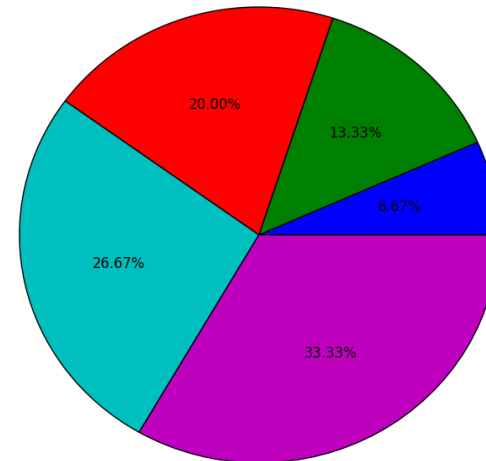
'%.4f%%' : 4 chiffres ..... (15,0000%)

# ILLUSTRATION

```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5], autopct='%f%%')  
plt.show()
```



```
import matplotlib.pyplot as plt  
plt.pie([1, 2, 3, 4, 5], autopct='%2f%%')  
plt.show()
```



# DISTANCE DES LÉGENDES

- **pctdistance**

Distance du texte **pourcentage** par rapport au centre. Par défaut à 0,6.

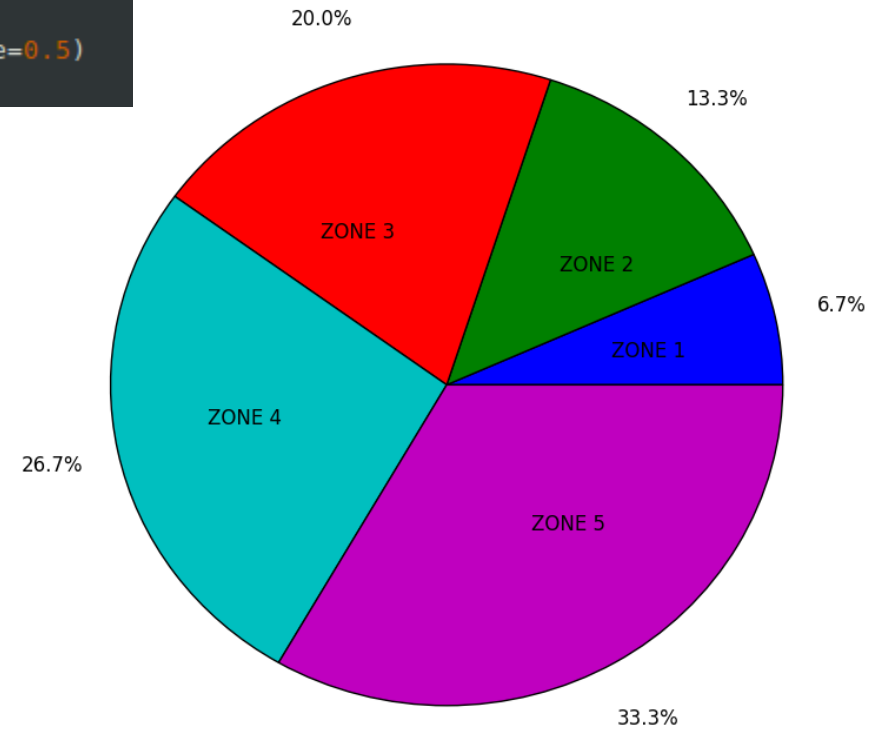
- **labeldistance**

Distance du texte **label** par rapport au centre. Par défaut à 1,1.



# OBSERVEZ

```
import matplotlib.pyplot as plt
labels = ["ZONE "+str(i) for i in range(1, 6)]
plt.pie([1, 2, 3, 4, 5], autopct='%.1f%%', labels=labels, pctdistance=1.2, labeldistance=0.5)
plt.show()
```



# BONUS

Voici un exemple de diagramme circulaire, légendé et coloré. Quelques paramètres sont utilisés mais il y en a bien d'autres. Vous pouvez vous en servir comme base pour réaliser le votre.

```
import matplotlib.pyplot as plt

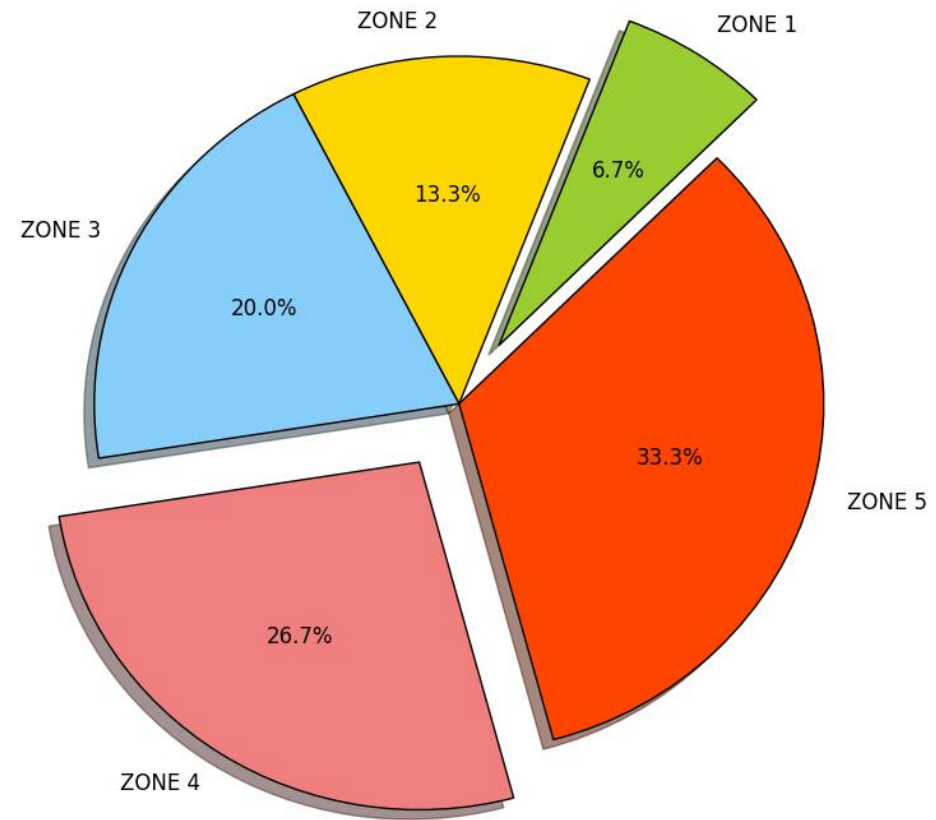
labels = ["ZONE "+str(i) for i in range(1, 6)]

plt.pie(
    [1, 2, 3, 4, 5],
    explode = [0.2, 0, 0, 0.2, 0],
    autopct = '%.1f%%',
    labels = labels,
    shadow = True,
    colors = ["yellowgreen", "gold", "lightskyblue", "lightcoral", "OrangeRed"],
    startangle = 45
)

plt.title("Diagramme circulaire")

plt.show()
```

Diagramme circulaire



# Subplot

Ou en français :  
plusieurs graphes dans une même fenêtre



Oui c'est possible !

Tous les graphes que vous avez vu jusqu'à maintenant sont sur des fenêtres individuelles. Mais il est possible d'afficher plusieurs graphes dans une seule et même fenêtre !

Attention tout de même à ne pas en afficher trop, sinon ce serait illisible.



# COMMENT QU'ON FAIT ?

Avec **`plt.subplot(n, m, a)`** ou **`plt.subplot(nma)`**.

Cette fonction va créer dans votre fenêtre une sorte de matrice, de dimensions  $n \times m$  ( $n$  lignes,  $m$  colonnes), vide pour l'instant, puisque que vous n'avez pas « affiché » vos graphes. Il vaut mieux prendre la première notation si vos  $n$  et  $m$  et  $a$  sont constitués de 2 chiffres ou plus.

Le dernier élément  $a$ , représente l'endroit dans la matrice où le graphe (`plt.plot()`) qui suit la commande `plt.subplot()` doit être placé. C'est une valeur entière.

Vous l'aurez donc deviné, vous devez appeler `plt.subplot()` avant et autant de fois que vous aurez de graphes à placer puis afficher.

Note : vous pouvez afficher des « graphes courbes » et des diagrammes en barres dans la même fenêtre.

1	2
3	4

## CHAQUE GRAPHE À SA PLACE

Prenez une page de bande dessinée « française » (pas japonaise). Dans quelle sens lisez vous les illustrations ? De gauche à droite puis de bas en haut. C'est le même principe. Exception pour

Si votre subplot() est de dimension 2x2, (donc 4 places), 1 correspond à la case en haut à gauche, 2 celle à droite, 3 en bas à gauche, et enfin 4 celle à droite. C'est tout simple !

Vous pouvez également étendre un graphe sur 2 emplacement (ou plus).

Attention ! C'est assez compliqué ! Un exemple serait plus explicatif que du texte.

1	
3	4

1	2
3	

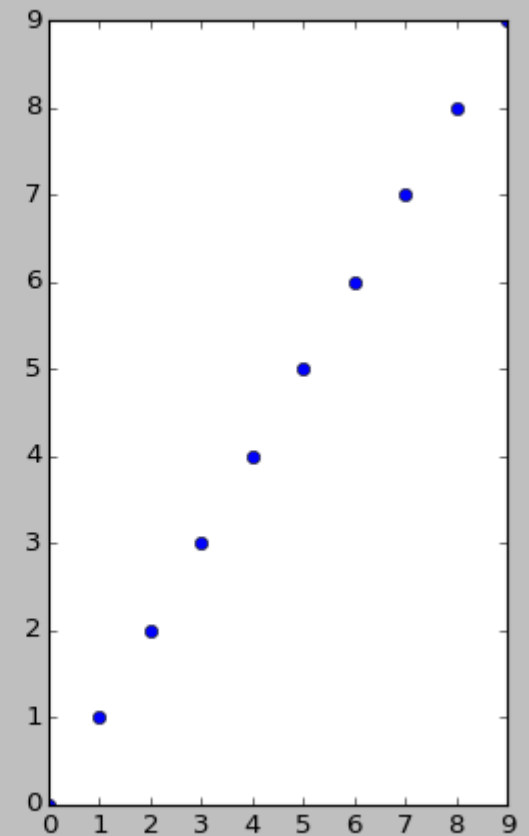
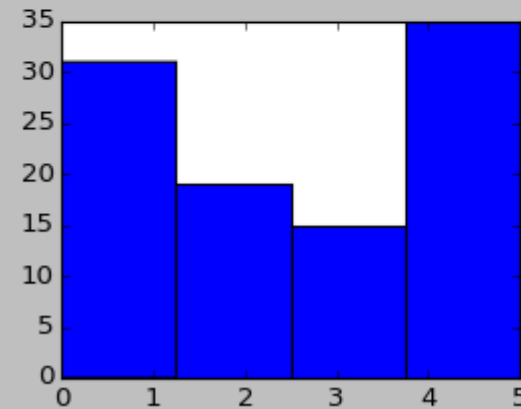
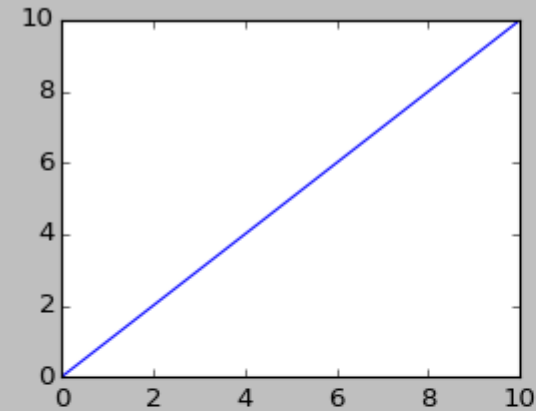
# OBSERVEZ ET CONTEMPLÉEEEEEZ !

```
plt.subplot(2,2,1)
plt.plot([0, 10], [0,10])

plt.subplot(2,2,3)
plt.hist([random.randint(0, 5) for i in range(100)], 4)

plt.subplot(122)
plt.plot(range(10), range(10), "o")

plt.show()
```



# EXPLICATIONS

Pour les deux graphes de gauche, rien de compliqué. Ils sont respectivement aux emplacements 1 et 3 de la fenêtre de « dimensions » 2x2. Il reste donc les places 2 et 4, là où est le troisième graphe.

J'ai utilisé **plt.subplot(122)** (ou (1,2,2)). Cela signifie que je place le graphe à la deuxième colonne, dans une fenêtre de dimension une ligne, deux colonnes.

Initialement, la fenêtre est composée de deux lignes, deux colonnes. En utilisant (1,2,2), le graphe est placé dans toute la colonne spécifiée, peu importe le nombre de lignes (une ligne équivaudra donc à deux lignes ici).

Essayez donc de placer les deux petits graphes de gauches sur la première ligne, et le troisième entièrement sur la deuxième ligne (ou inversement). Y arriverez-vous ? La solution est à la diapo suivante. Essayez de réfléchir avant de voir la solution.



# SOLUTION

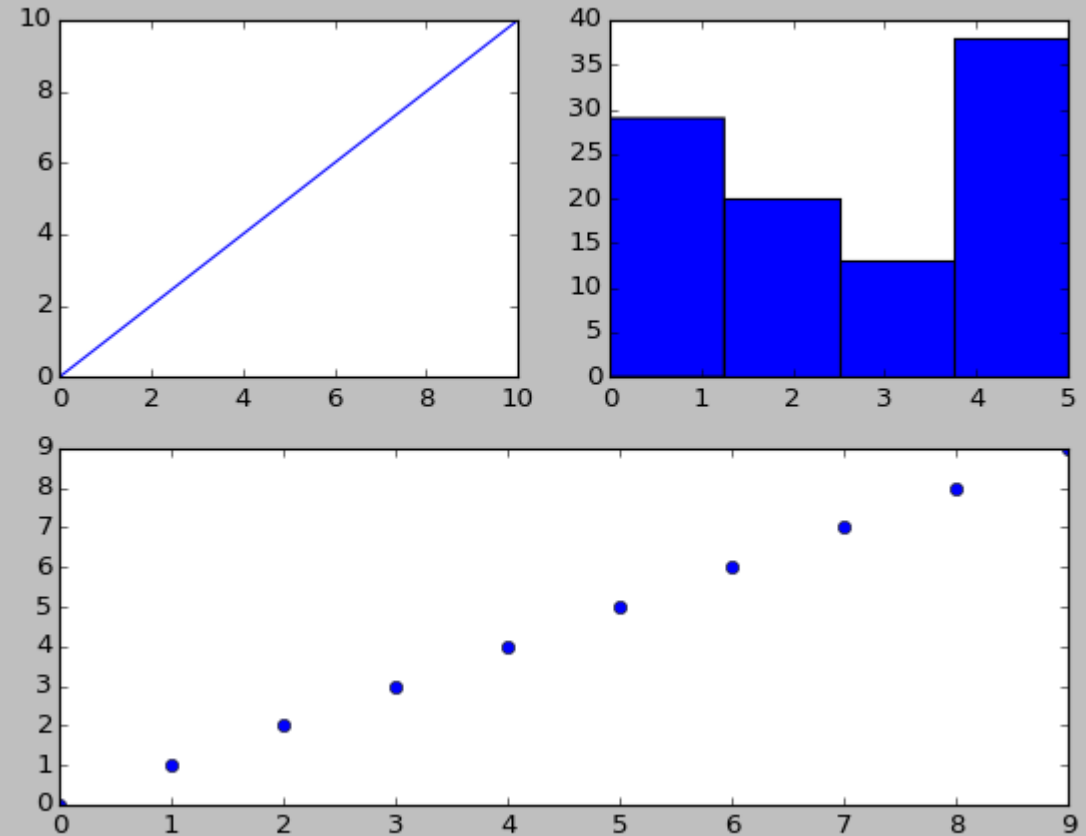
```
plt.subplot(221)  
plt.plot([0, 10], [0, 10])
```

→ 

```
plt.subplot(222)  
plt.hist([random.randint(0, 5) for i in range(100)], 4)
```

→ 

```
plt.subplot(212)  
plt.plot(range(10), range(10), "o")  
  
plt.show()
```





# LIEN AVEC NUMPY

Numpy est un module qui permet de manipuler les matrices. Il contient également d'autres fonctions diverses très utiles.

Le module **pylab** regroupe matplotlib et numpy. Il contient également un module nommé « scipy », mais ce dernier ressemble très fortement à numpy.

Si vous utilisez numpy et matplotlib à la fois, vous pouvez importer **pylab**. Mais cela est plutôt déconseillé. En effet, dans plusieurs cas, des fonctions identiques de nom peuvent se trouver dans deux modules différents. Lorsque vous appelez cette fonction, vous ne savez pas de quelle module la fonction a été prise. D'un module à un autre, une fonction identique de nom peut donner des résultats différents (1 chiffre en plus par exemple, ou en moins).

C'est pour cela qu'il est conseillé d'importer les modules numpy et matplotlib séparément, mais libre à vous de faire ce que bon vous semble.

Vous savez maintenant comment tracer de belles courbes et de belles barres pour épater les jolies filles ! (Et garçons). Toutes les informations qui se trouvent dans ces diapositives sont selon moi les plus importantes et les plus courantes. Il existe bien sûr pleins d'autres améliorations possibles de vos graphes, et pour cela, je vous laisse vous documenter !

Voici le lien du site officiel du module matplotlib d'où j'ai tiré la plupart des informations des diapos :

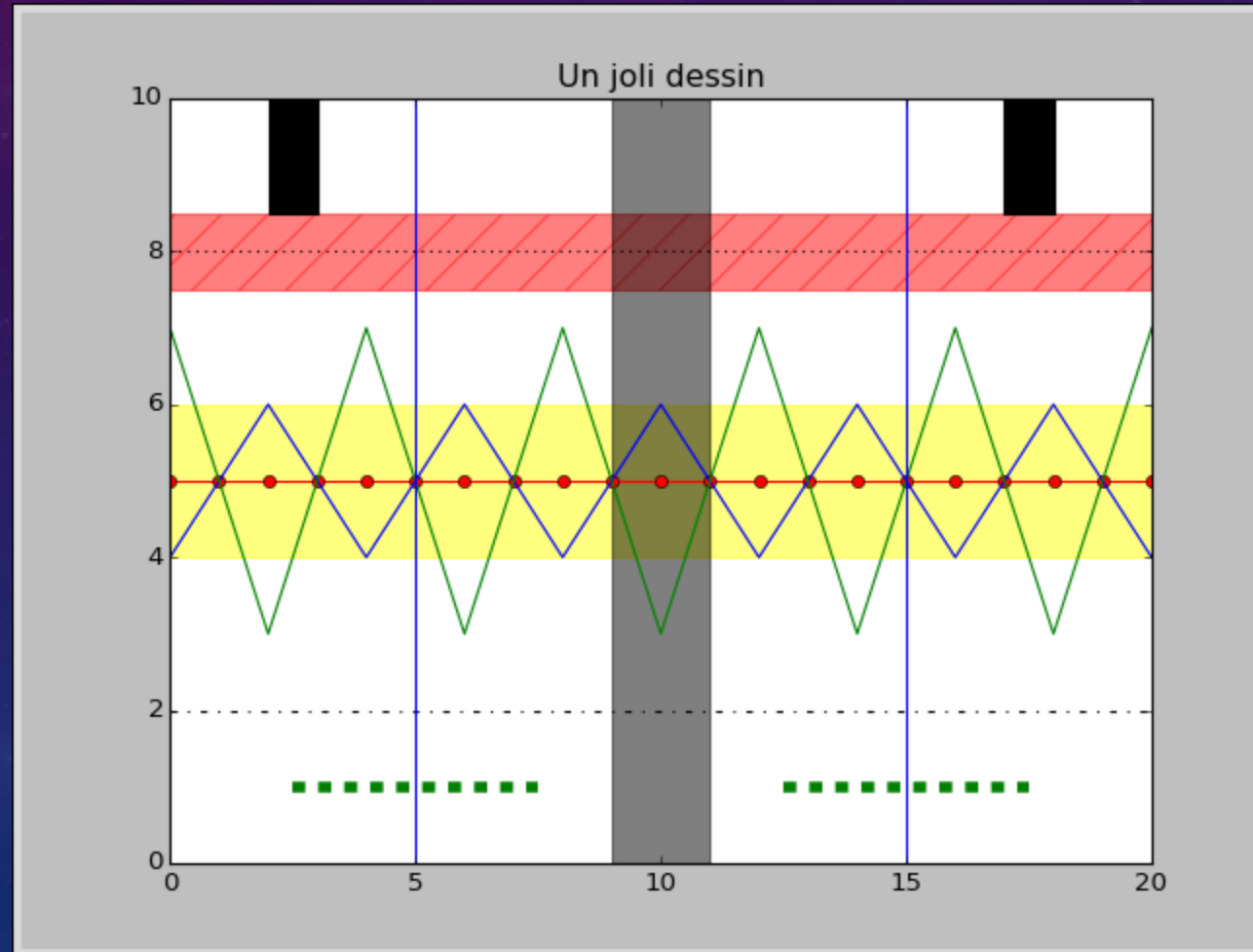
<http://matplotlib.org/>

En anglais malheureusement (ou heureusement). Il existe pleins d'autres sites web sur le sujet (en français !), mais je vous laisse chercher !

Si vous avez apprécié les diapositives mettez un pouce bleu juste en bas de l'écran, ou un « j'aime » ... Nan je déconne !

# EXERCICES !

- 1) Tracer 5 courbes de 5 couleurs différentes et de 5 styles différents sur le même graphe (veillez à ce qu'elles soient bien distinctes).
- 2) Reproduire « approximativement » (ou parfaitement) le « graphe » suivant :



3) Voici les valeurs de températures moyennes (en degrés,  $(\text{max}+\text{min})/2$ ) au cours de plusieurs années. Tracer l'évolution de la température en fonction du mois pour toutes les années.

Année	Janvier	Février	Mars	Avril	Mai	Juin	Juillet	Août	Septembre	Octobre	Novembre	Décembre
2011	5,4	7,1	10	15,8	17	18,8	18,5	20	19	14,2	10,5	7,9
2012	7,2	3	11,5	10,3	16,3	17,7	19,6	21,7	16,7	13,1	8,2	6,9
2013	4,2	3,7	6,1	11,2	12,9	17,5	22,9	20,8	17,5	14,8	7,9	6,8

4) Le tableau suivant représente le niveau de précipitation (en mm) au cours de l'année 2013. Visualiser à l'aide d'un diagramme en barres le niveau de précipitation pour chaque mois. Optionnel : Sur le même graphe, tracer la courbe reliant les sommets de chaque barre.

	Janvier	Février	Mars	Avril	Mai	Juin	Juillet	Août	Septembre	Octobre	Novembre	Décembre
2013	35,2	37,4	26,4	10,6	111,2	64	48,2	37	49,1	43,4	69,7	29



# EXERCICES (SUITE)

5) Soit une grille de loto (1-49). Lorsque vous jouez au loto, soit vous cochez vous-même les numéros parmi ceux proposés (on omettra les numéros chances/joker ici), soit vous demander un « flash ». Ce dernier vous donne une série aléatoire de numéros parmi les 49 de votre grille (chaque numéro a une probabilité de  $1/49$  de sortir). On fixe à 5 le nombre de numéros cochés pour une personne.

Simuler 1000 flashes pour 1000 personnes jouant au loto et créer une visualisation des effectifs de chaque numéro sous forme d'histogramme.

6) Optionnel : le tirage gagnant est le suivant : 1, 30, 15, 21, 48 (vous pouvez choisir vous-même le tirage gagnant de votre choix, avec un 1001-ème tirage par exemple).

Il y a t-il un gagnant ?



7) Voici les ingrédients d'une pizza simplifiée et leurs proportions

- Pâte jaune (50%)
- Sauce tomate rouge (30%)
- Fromage orange (10%)
- Jambon rose (5%)
- Champignons blancs (5%)

Représentez votre pizza « décomposée » sous forme d'un diagramme circulaire, légendes incluses.

8) Le tableau ci-dessous représente les 5 marques ayant vendu le plus de voitures en 2017 en France.

Marque	Renault	Peugeot	Citroen	Volkswagen	Dacia
Ventes	416577	366872	201373	139360	117865

À l'aide d'un diagramme circulaire, représentez la part de vente de chaque marque, légendes incluses.