I) Présentation des bibliothèques

Rappels : Une bibliothèque est un fichier python (habituel) qui contient des lignes de code définissant un certain nombre de fonctions. On l'importe à l'aide des commandes

`import` (nom de la bibliothèque)
ou : `import` (nom de la bibliothèque) `as` (abréviation)

Exemple :
```
import random
import random as rd
```
Dans le premier cas on appellera la fonction `gauss` avec la syntaxe `random.gauss`, dans le second avec `rd.gauss`.

A) Bibliothèque `random`

La bibliothèque `random` donne des outils pour générer des nombres au hasard : la meilleure approximation possible, pour un ordinateur, d'un nombre « tiré au hasard » s'appelle un nombre *pseudo-aléatoire*.

- `random.random()` : retourne un float pseudo-aléatoire dans l'intervalle $[0.0, 1.0[$.
- `random.gauss(mu,sigma)` : retourne un nombre au hasard selon la loi gaussienne. mu est la moyenne et sigma l'écart type.
- `random.randrange(start, stop[, step])` : retourne un élément au hasard dans la liste `range(start, stop, step)`.
- `random.randint(a,b)` raccourci pour `random.randrange(a,b+1)` : retourne un entier N au hasard entre a et b ($a \leqslant N \leqslant b$).

Une liste exhaustive des fonctions se trouve sur `http://docs.python.org/3/library/random.html` avec des exemples.

Exercice 1
Faire afficher une liste de 5 float obtenu avec la commande `random`.

B) Bibliothèque `matplotlib`

La bibliothèque `matplotlib` donne des outils pour faire des dessins en 2D. On peut faire aussi bien des graphes de fonctions, des histogrammes que des choses plus savantes (de nombreux exemples sur `http://matplotlib.org/gallery.html`).
C'est une bibliothèque dont nous allons beaucoup nous servir pour faire des représentations graphiques.

- `matplotlib.pyplot.plot(x,y)` : Prend deux listes et trace la courbe (affine par morceaux) passant par les (x_i, y_i). C'est la méthode naïve pour tracer une courbe : on place les points et on les relie.
- `matplotlib.pyplot.scatter(x,y,c='b',marker='o',s='20')` : place une série de points (sans les relier) en (x_i, y_i). La variable `c` règle la ou les couleurs utilisées ; `marker` le symbole utilisé pour marquer le point ; `s` la taille du symbole. Cf exercice 3.
- `matplotlib.pyplot.hist(L,nombre)` : `L` est une liste qui contient les données. C'est le seul argument obligatoire. Le nombre contient le nombre de barres de l'histogramme.
  On peut rajouter l'argument `normed='True'` pour avoir des proportion plutôt que des effectifs (cf exercice 2).

Comment voir et sauver ces merveilleuses figures ?

- `matplotlib.pyplot.show()` : Affiche toutes les figures construites jusqu'ici. En mode non interactif (i.e. lorsqu'on lance un programme tapé dans l'éditeur), l'affichage des figures bloque l'exécution du programme, jusqu'à ce que toutes les figures aient été fermées.
- `matplotlib.pyplot.savefig(nom de fichier)` : Sauve la figure courante. Le nom du fichier est donné sous forme de `string`, et l'extension indique le format de sauvegarde.
  Exemple : `matplotlib.pyplot.savefig('Ma_jolie_figure.png')`.
- `matplotlib.pyplot.clf()` : comme CLear Figure, efface la figure courante (sans cette instruction, tout s'empile sur le même graphique).

Plus de doc : `http://matplotlib.org/api/pyplot_api.html`

Exercice 2 (prise en main histogrammes)

1) Tester le code suivant :

```
import matplotlib.pyplot as plt
import random
L = [random.random() for i in range(10000)]
plt.hist(L,20)
plt.show()
```

2) Modifier puis supprimer complètement le `20` : que se passe-t-il quand on ne rentre pas de valeur ? C'est ce qu'on appelle le « comportement par défaut ». Quand un argument n'est pas obligatoire, c'est qu'il y a une valeur par défaut si cet argument n'est pas précisé.

3) Rajouter `,normed='True'` après le `20`. Que se passe-t-il ?

4) Sauvez la figure à l'aide de la commande `savefig`.

II) Applications

Exercice 3

1) Comprendre puis taper le programme suivant :

```
import matplotlib.pyplot as plt
import random
x, y = [], []
for i in range(10000):
    a, b = 2*random.random()-1, 2*random.random()-1
    x.append(a)
    y.append(b)
xyc = range( len( x ) )
plt.scatter(x,y,c =xyc, marker = '.', s=200)
plt.axis('equal')
plt.show()
```

2) On va utiliser `random` pour calculer des décimales de π : proposez une méthode. Puis proposez un algorithme, en python.

3) Illustrez à l'aide d'une figure. On pourra faire se succéder deux `scatter`.

Exercice 4 (Courbes)

On commence par stocker des données numériques sous forme de listes (plus exactement de tableaux `numpy`, on y reviendra) contenant 256 vaeurs entre $-\pi$ et π (fermé).

```
import numpy as np
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
```

1) Essayer les commandes `plt.plot(X, C)` et `plt.plot(X, S)`. Faire afficher le graphique créé.

2) Axes : comprendre les commandes :

```
plt.xlim(-4.0, 4.0)
plt.xticks(range(-4,9))
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
```

Trouver les commandes analogues pour l'axe des y.

Pour avoir des axes dans un style plus conventionnel :

```
ax = plt.gca()  # gca signifie: 'get current axis'
ax.spines['right'].set_color('none') #l'axe de droite est effacé (couleur none)
ax.spines['top'].set_color('none') #idem pour l'axe du haut
```

```
ax.xaxis.set_ticks_position('bottom') #on ne met que des étiquette sur
                                      #l'axe du bas
ax.spines['bottom'].set_position(('data',0))#et l'axe du bas est placé en 0
ax.yaxis.set_ticks_position('left') #idem pour l'axe de gauche.
ax.spines['left'].set_position(('data',0))
```

3) Tracé : faire varier les options `color`, `linewidth`, `linestyle`.
Rajouter une légende sur les axes : options `xlabel` et `ylabel`.

4) Pour les amateurs de raffinement : des annotations

```
t = 2 * np.pi / 3
plt.plot([t, t], [0, np.cos(t)], color='blue', linewidth=2.5, linestyle="--")
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')

plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, np.sin(t)), xycoords='data',
             xytext=(+10, +30), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t, t],[0, np.sin(t)], color='red', linewidth=2.5, linestyle="--")
plt.scatter([t, ],[np.sin(t), ], 50, color='red')

plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$',
             xy=(t, np.cos(t)), xycoords='data',
             xytext=(-90, -50), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```

et du tuning des étiquettes sur les axes :

```
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(16)
    label.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.65))
```

D'après scipy-lectures.github.io

Exercice 5 (Marches aléatoires)
Une marche aléatoire (en dimension 2) est une suite de points, où chacun est obtenu par un saut indépendant à partir du point précédent (et rien de plus : le système « perd la mémoire » à chaque saut).
Dans l'exemple étudié, on autorise les saut de longueur 1 sur l'axe des x (donc $+1$ ou -1 sur x) ou (exclusif) sur l'axe des y (donc $+1$ ou -1 sur y), de façon équiprobable.

1) Créer, à l'aide d'une fonction, deux listes, `x_liste` et `y_liste` contenant les coordonnées `(x[i],y[i])` des points d'une marche aléatoire de N pas.

2) Faire afficher la marche avec `scatter` et un code couleur, puis avec `plot`.