Informatique - TD 3 | Rappels sur les graphiques et bibliothèques

Rappel : *Pour faire des graphiques en Python, il est nécessaire de charger une bibliothèque. Pour les graphiques "basiques", on utilise généralement* ***matplotlib.pylab*** *ou* ***matplotlib.pyplot*** *. Une documentation complète (mais en anglais...) est disponible sur le site* http ://matplotlib.org/*. Vous y découvrirez les nombreuses possibilités de ces bibliothèques (Histogrammes, diagrammes, barres d'erreurs, boites à moustaches et beaucoup d'autres). Pour remarque : il n'y a pas de fichier Python à disposition pour ce TP.*

Petit point sur l'utilisation d'une bibliothèque

Si l'on souhaite charger toutes les commandes d'une même bibliothèque, on a essentiellement deux façons de faire. Voyons par exemple avec `matplotlib.pyplot` :

```
from matplotlib.pyplot import * # les fonctions de
    la bilbiothèque seront utilisables sans préfixe
     (Technique DECONSEILLEE, cf ci-dessous.)
```

ou

```
import matplotlib.pyplot as plt # l'utilisation des
    fonctions nécessitera le préfixe plt.
    # (Technique conseillée, cf ci-dessous.)
```

Si la deuxième manière d'importer peut sembler a priori plus compliquée à mettre en oeuvre, c'est donc néanmoins celle qui est À PRIVILEGIER !

En effet, elle permet entre autre :
- après quelques utilisations, de ne pas oublier de quelle bibliothèque proviennent les commandes utilisées.
- en cas d'utilisation de plusieurs bibliothèques, si une fonction de même nom est présente dans au moins deux des bibliothèques (oui, c'est possible !), de savoir laquelle exactement on utilise. Prenons un exemple avec la fonction `cos` présente dans `math` et `numpy`. Tapez les commandes et compilez le texte suivant :

```
1  from math import *
2  from numpy import *
3
4  A=cos(0)
5  B=cos([0,pi/2])
```

Observez que `A` est une valeur réelle et `B` est un tableau (’`array`’) contenant $\cos 0$ et une approximation de $\cos \frac{\pi}{2}$.
Échangez maintenant les deux premières lignes et compilez à nouveau.

```
1  from numpy import *
2  from math import *
3
4  A=cos(0)
5  B=cos([0,pi/2])
```

Observez le résultat. Vous obtenez... un gros paragraphe rouge. Notez qu'il y est écrit : `B=cos([0,pi/2]) TypeError: a float is required` ce qui signifie que Python attendait ici un réel (`float`) et non une liste.

Il faut donc être conscient que les fonctions introduites par une nouvelle bibliothèque sous la forme "`from ... import *`" écrase toutes celles déjà présentes ayant le même nom. Ainsi, si la fonction cos de `numpy` permet la gestion de listes, celle de `math` ne le permet pas. Essayez maintenant

```
1  import numpy as np
2  import math as mt
3
4  A=mt.cos(0)
5  B=mt.cos(mt.pi/2)
6  C=np.cos([0,np.pi/2])
```

Plus aucun problème, même si `math` a été introduite après `numpy`.

Remarque : ***plt*** *,* ***np*** *et* ***mt*** *sont les noms que l'utilisateur a choisi ici pour les préfixes des fonctions. Vous pouvez le changer à votre guise (essayez) mais sachez qu'il s'agit ici des préfixes usuellement utilisés et qu'ils auront l'avantage d'être compris par la plupart des utilisateurs de Python, notamment des correcteurs et du jury ... !*

On peut également se passer du " ***as ...*** *", c'est-à-dire par exemple écrire tout simplement*

```
import numpy
```

Dans ce cas, le préfixe sera ***numpy*** *et non* ***np*** *.*

Rappels et compléments sur les outils graphiques de base

Rappel : *On rappelle que pour Python, un graphique est un ensemble de points (abcisses + ordonnées) reliés entre eux dans un ordre précis. La fonction de base pour des graphiques "simples" est* **`plot`** *, mais voici pour compléter quelques commandes utiles :*

```
import matplotlib.pylab as plt
# ou
import matplotlib.pyplot as plt

plt.plot(X,Y)        # créer la suite de segments (ligne brisée) reliant les points (X[i],Y[i]) (dans l'ordre énoncé).
plt.scatter(X,Y)     # crée l'ensemble des points
                     # (X[i],Y[i]) sans les relier
plt.show()       # permet d'afficher le graphique
plt.close()      # permet de fermer la fenêtre graphique en cours sans cliquer sur le bouton ''fermeture''
plt.close("all")# permet de fermer toutes les fenêtres graphiques

plt.legend()         # met en place une légende
plt.title("titre")   # donne un titre au graphique
```

Pour approfondir un peu tout ceci, vous pouvez toujours faire appel à **`help(...)`** *en mettant le nom de votre fonction à la place des ... ou consulter une rubrique d'aide sur internet.*
On rappelle que la fonction **`plot`** *admet deux arguments de base notés ici* **`X`** *(les abscisses) et* **`Y`** *(les ordonnées) qui s'utilisent de la façon suivante :*
`plot(X,Y)` *relie* ***dans l'ordre*** *les points* $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ *si* **`X`**$=[x_0, \ldots, x_n]$ *et* **`Y`**$=[y_0, \ldots, y_n]$*. Remettons ceci en pratique :*

Exercice 1:

1. Tapez les commandes suivantes et voyez si c'est cohérent avec ce que vous attendez en regardant ce qui est demandé à Python.

```
1  X=[1,1]
2  Y=[0,3]
3  plt.plot(X,Y)
4  plt.show()
```

2. Tapez maintenant les commandes suivantes et demandez-vous également si c'est cohérent avec ce que vous attendiez

```
1  import numpy as np  # à taper seulement si la biltiothèque n'a pas déjà été importée auparavent, ce qui est normalement déjà fait...
2  plt.close("all")
3  X=np.linspace(0,10,6)   # X : liste de 6 nombres équitablement répartis entre 0 et 10 (inclus)
4  print('X=',X) # on vérifie que X est bien la liste annoncée
5  Y= np.cos(X)
6  plt.plot(X,Y)
7  plt.show()
```

3. Modifiez maintenant le "nombre de points" de la fonction `linspace` (i.e. troisième argument) en remplaçant par exemple 6 par 100. Voyez la différence et demandez vous alors pourquoi on a cette fois-ci une impression de "courbe".

Voici un rappel de quelques options pratiques à insérer dans `plot` *(On les réessaiera dans l'exercice suivant, donc pas la peine de les taper maintenant hors contexte)* :

```
label="..."     # Donne un nom à la courbe. Ce nom apparaîtra dans la légende si on y fait appel.
color='...'     # couleur de la courbe.
marker='...'    # apparence des points.
linestyle='...' # apparence de la ligne.
linewidth='...' # épaisseur du trait
```

une petite option pratique à insérer dans `legend` :

```
loc='...'   # Permet d'imposer une localisation forcé pour placer la légende sur le graphique.
loc='best' # Place la légende à l'emplacement jugé le meilleur par python.
```

et pour finir, des options de gestion d'axes :

```
plt.xlim(a,b)   # permet de visualier les axes de l'abcisse a à l'abcisse b si a<b (et de retourner le graphique horizontalement si b<a !!)
plt.ylim(a,b)   # même chose avec les ordonnées
plt.axis('equal')   # permet d'othonormaliser les axes (i.e. les mettre à la même échelle.)
```

Exercice 2: On repasse en revue ces commandes

1. Tapez maintenant le programme suivant et voyez ce que ça donne. Retirer ensuite la ligne 10 (`plt.legend()`) et voyez ce que ça change.

```
1  plt.title('Essai')
2
3  plt.close('all')
4  X=[1,1]
5  Y=[0,3]
6  plt.plot(X,Y,label='c1',linewidth='3')
7
8  X=[1,1,2]
9  Y=[0,3,5]
10
11 plt.scatter(X,Y,label='c2))
12
13 Y=[x**2 for x in X]
14 plt.plot(X,Y,linestyle='dashed')
15
16 plt.legend()
17 plt.show()
```

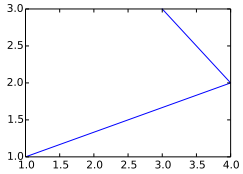
2. On pourra ensuite rajouter dans un des `plot` par exemple l'option `color='green'` (ou toute autre couleur de votre choix) pour observer son effet.

3. On pourra également rajouter dans la commande `legend` l'option `loc='center right'` pour observer là encore ce qui est modifié. Pour rappel, si on veut consulter toutes les localisations possibles de la légende, on peut taper `help(plt.legend())` dans le shell (constater qu'il existe d'ailleurs une très longue liste d'options pour cette commande) et aller à l'option `loc`. *Il y a 11 possibilités de localisations différentes que vous pouvez tester si vous en avez le temps.*

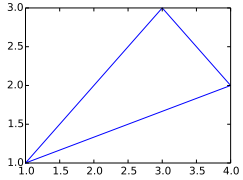
4. Pour finir, testez un peu les commandes de gestion d'axes `xlim, ylim et axis`. (On pourra notamment demander par exemple `plt.xlim(-2,10)` puis `plt.xlim(10,-2)` pour voir l'effet de l'invertion des bornes. De même pour `ylim`.)

Exercice 3: Tracé de lignes, vous même cette fois ci !

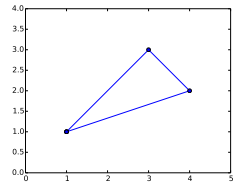
1. Tracer sur une courbe la ligne brisée démarrant au point $(1,1)$ et arrivant au point $(3,3)$ en passant par le point $(4,2)$. On doit obtenir ceci :



2. Après avoir fermé la fenêtre de la courbe précédente, tracer maintenant le triangle complet formé par les trois points précédents. (On demande de faire le tracé complet à l'aide **d'un seul** `plot`). On doit obtenir ceci :



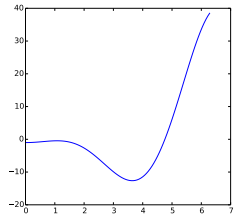
3. Trouver maintenant le moyen de rajouter les sommets du triangle en leur associant un point et "écarter" le bord de la fenêtre graphique afin de mieux visualiser le triangle avec les points ainsi formés (on pourra pour ceci penser aux commandes `xlim` et `ylim`). On doit obtenir ceci :



Exercice 4: Tracé d'une courbe représentative de fonction

Comme vu en début de TP, en diminuant de façon pertinente l'écart entre les points, on peut donner l'illusion d'une courbe.

1. Tracer la courbe de la fonction $f : x \mapsto x^2 \cos x - 1$ sur l'intervalle $[0; 2\pi]$. Vous devez obtenir ceci :



2. Sur le même graphique, rajoutez un titre.

Exercice 5: Les graphiques à paramètre / faire plusieurs graphiques sur le même dessin

1. Tracer un cercle de centre $(0,0)$ et de rayon 1. *(On rappelle que ce cercle est formé de points de coordonnées* $(\cos\theta, \sin\theta)$ *avec* θ *variant dans un intervalle de longueur* 2π*.)*
   On pourra éventuellement rajouter `plt.axis('equal')` pour avoir un rendu visuel avec des axes orthonormés. (Donc, un vrai cercle.)

2. Dessiner maintenant à l'intérieur de ce cercle un carré dont :
   - les sommets sont sur le cercle
   - le premier point est pris au hasard par Python sur le cercle. (Pour ce faire, vous disposez de la commande `rd.random()*2*np.pi` qui, une fois les bibliothèques `random` et `numpy` importées en tant que `rd` et `np`, vous rend un angle aléatoire compris entre 0 et 2π.)

On constate qu'avec les commandes revues jusque là, si on voulait visualiser deux graphiques en même temps, il fallait nécessairement les superposer sur un même dessin. Or ceci n'est pas toujours pertinent (si les deux courbes ou dessins n'ont aucun rapport). Nous allons (re ?)voir ici comment obtenir plusieurs graphiques indépendants.

Exercice 6: Avoir plusieurs fenêtres contenant chacune un graphique

```
1  plt.figure(1) # démarre une première figure. On
       fait son dessin à la suite comme d'habitude. La
       fenêtre d'affichage s'appellera "figure 1". A
       mettre absolument avant le plot de la figure en
       question !!
2
3  plt.figure(2) # démarre une autre fenêtre graphique
       .
4  # etc...
5  plt.show() # affiche les deux fenêtres.
```

1. Afficher les deux graphiques de l'exercice 2 (fonction) et 3 (carré inscrit dans un cercle) sur deux fenêtres distinctes.

2. Écrit ainsi, les deux fenêtres portent comme nom "figure 1" et "figure 2". Si on veut modifier ceci, il suffit de mettre un titre à la place des nombres dans `plt.figure(...)`. Testez sur les figures de la question précédente de manière à leur donner les noms respectifs "fonction f" et "Carré inscrit".

Exercice 7: Avoir deux dessins côte à côte dans une même fenêtre.

Pour ceci, nous allons utiliser la commande `subplot` de `plt` .

Tapez les commandes suivantes (ou compilez directement depuis le fichier donné) :

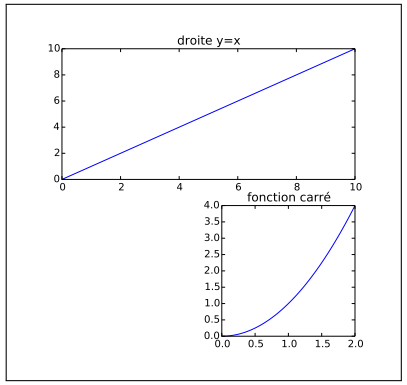
```
1  plt.figure('cote à cote') # nom de la fenêtre
       graphique
2  plt.clf()
3
4  plt.subplot(121) # premier graphique, à gauche
5
6  X=[0,10]
7  Y=[0,10]
8  plt.plot(X,Y)
9
10 plt.subplot(122) # deuxième graphique, à droite
11
12 X=np.linspace(0,2,100)
13 Y=[x**2 for x in X]
14 plt.plot(X,Y)
15 plt.show()
```

1. Changer les valeurs $1, 2$ par d'autres chiffres dans les subplot et observer les résultats.

En fait, ***`subplot(xyz)`*** *découpe la zone graphique sous la forme d'un quadrillage invisible avec* x *cases en hauteur et* y *cases en largeur (donc au total* $x \times y$ *cases.)* ***`z`*** *est ensuite le numéro de la case dans lequel on souhaite mettre le graphique, sachant que les cases sont numérotées de gauche à droite et de haut en bas, en commençant par le n° 1. Par exemple, pour* $x = 2, y = 3$*, le graphique est découpé comme suit et les numéros indiqués correspondent au "*z*" :*

1	2	3
4	5	6

2. Réécrire les quelques lignes ci-dessus de manière à obtenir le résultat suivant : (constatez qu'on a déplacé les graphiques, mais également mis des titres...)



Exercice 8: Nuage de points *(Bonus : entrainement si vous avez fini le reste)*

Dans ce problème, on supposera la bibliothèque `numpy` chargée avec le raccourci `np`. On utilisera de plus les fonctions suivantes :

```
np.random.normal(mu,sigma) # simule l'obtention d'
    un nombre aléatoire suivant une loi normale d'
    espérance mu et d'écart type sigma.
plt.scatter(X,Y) # avec éventuellement les options
    s= et alpha= qui seront expliqués plus tard
```

(La loi normale sera (re)vue un peu plus tard dans l'année. L'objet n'est pas ici de comprendre le résultat obtenu par la fonction `normal` *mais simplement de faire les graphiques demandés afin de pouvoir utiliser ce nuage de points par la suite.)*

D'après diverses données collectées sur internet, une femme en France pèserait en moyenne actuellement 67kg pour une taille de 1,64m. On estime l'écart type de la taille à 6cm.
Une homme quant à lui pèserait en moyenne 82kg pour une taille de 1,77m. On estime l'écart type de la taille à 5,5cm.
Dans les deux cas (hommes et femmes), les données sur l'écart type du poids sont relativement inexistantes. Pour les besoins de l'exercice, nous allons donc estimer ceci dans les deux cas à 4 kg.

On va considérer ici 100 femmes et 100 hommes dont les données sont construites de manière aléatoire de la façon suivante :

1. **a)** Créer une liste `TailleFemme` de 100 tailles aléatoires suivant la loi normale d'espérance 164 et d'écart type 6.
b) Créer une liste `TailleHomme` de 100 tailles aléatoires suivant la loi normale d'espérance 177 et d'écart type $5,5$.

2. **a)** Créer une liste `PoidsFemme` contenant les 100 poids aléatoires des femmes en suivant la loi normale d'espérance 67 et d'écart type 4.
b) Créer une liste `PoidsHomme` contenant les 100 poids aléatoires des hommes en suivant la loi normale d'espérance 82 et d'écart type 4.

3. **a)** Dessiner le nuage de points avec en abscisse : la taille des femmes et en ordonnées : le poids des femmes.
b) Mettre sur le même graphique le nuage de point des hommes, dans une autre couleur.

4. Mettre une légende pour savoir quel nuage correspond aux hommes (et aux femmes)

5. **a)** faire une nouvelle liste avec l'IMC de chaque homme (et également celui de chaque femme). (*La formule de calcul est la suivante : "poids en kg/taille*2 (en m)".)*
b) Calculer l'IMC moyen pour les deux listes "hommes" et "femmes" sur les deux échantillons respectifs obtenus, ainsi que l'écart-type de chacune de ces listes.

Le but est maintenant de visualiser l'IMC sur le nuage, en ne rajoutant aucun point, mais en modifiant l'allure des points déjà présents au sens suivant : plus l'IMC est important, plus le point grossit.
Pour ce faire, l'option `s=` dans scatter permet de fixer la taille des points.

6. L'option `s=`*"un entier positif"* permet de modifier la taille du symbol utilisé. (La valeur par défaut est `s=20`). Testez cette option sur un des deux nuages, en modifiant plusieurs fois la valeur afin d'observer le phénomène. Voyez pour quelles valeurs la différence commence à être notable à l'oeil.

7. L'option `s=[liste de tailles]` permet de modifier la taille du symbol, mais point par point. Les valeurs doivent bien entendu être données dans le même ordre que pour les listes `X,Y`!
a) Faire un nouveau graphique, contenant les nuages "homme" et "femme", en modifiant la commande `scatter` de manière à ce que la largeur du point soit cette fois-ci proportionnelle à l'IMC de chacun. On pourra à cet effet se servir de l'option `s=[...]` dans `scatter` où `[...]` est la liste de toutes les tailles de points. Observez si on voit vraiment une différence entre les points. Expliquez pourquoi.
b) On propose de refaire ce graphique en prenant plutôt une taille propotionnelle à " $\max(IMC) - \min(IMC)$". Déterminer visuellement un coefficient de propotionnalité (identique pour les hommes et les femmes) adapté à votre écran, permettant de visualiser correctement la variation de l'IMC sur chacun des points. On mettra, comme avant, les nuages "homme" et "femme" sur le même graphique. Vérifiez si le résultat vous parait cohérent.
(Pour plus de visibilité, on pourra éventuellement rajouter une option `alpha= nombre entre 0 et 1` *qui rend les points plus ou moins transparents.)*