Guide de survie Python pour les TD capteurs UE1.2 - FISE 1A En cours de construction ...

Liste des FAQs (Frequently Asked Questions)

Q1 - Comment écrire et exécuter un programme Python en mode console simple ?

Le logiciel "IDLEX (Python GUI).exe" est la méthode la plus simple pour créer et exécuter un petit programme Python. Tapez la touche "Window" puis "py" ou "idlex", l'icône du logiciel "IDLEX (Python GUI)" devrait apparaître.
Si cela ne marche vous pouvez exécuter ce programme à partir d'un terminal (voir FAQ Q3)

Q2 - Comment écrire et exécuter un programme Python avec le logiciel Spyder ?

Spyder est un logiciel un peu plus complexe. Tapez la touche "Window" puis "spyd" , l'icône du logiciel "Spyder" devrait apparaître.
Si cela ne marche vous pouvez exécuter ce programme à partir d'un terminal (voir FAQ Q3)

Q3 - Comment exécuter un interpréteur Python (IDLEX ou Spyder) depuis un terminal windows ?

- Ouverture d'un terminal :
  en cliquant en bas à gauche sur l'icône "Windows" avec le bouton droit, puis cliquer sur le menu "Exécuter".
  Tapez "cmd" dans le champ "Ouvrir :" et cliquer "OK", le terminal doit apparaître
- Dans le terminal, pour lancer IDLEX ou Spyder, entrez l'une des deux commandes suivantes (vous pouvez utiliser la touche tabulation pour ne pas tout taper ou utiliser le copier-coller) :
  Pour IDLEX :
  `"C:\applis\WinPython-64bit-3.6.1.0Qt5\IDLEX (Python GUI).exe"`
  Pour Spyder :
  `C:\applis\WinPython-64bit-3.6.1.0Qt5\Spyder.exe`
- Ensuite, utiliser les menus pour , soit créer un nouveau programme, soit enregistrer votre programme dans un fichier, soit récupérer le fichier d'un programme existant.
- Pour exécuter le programme, cliquer sur "Run", puis "Run module" (ou tapez simplement le raccourci clavier : touche F5)

Q4- Comment écrire et exécuter un programme Python avec le logiciel PyCharm ?

Attention, à la rentrée 2019 (FISE 2022) , PyCharm n'est pas bien configuré sous Windows sur tous les PC de l'école. Si vous souhaitez utiliser PyCharm, il faudra le faire sous Linux.
Utilisez PyCharm. Contrairement aux FAQs Q1 à Q2, il n'y a rien à faire. Ce logiciel, prévu pour les gros projets informatiques, n'est pas très bien adapté aux travaux demandés dans le cadre du cours "Capteurs et Systèmes de Mesure". Cependant si vous le maîtrisez bien , vous pouvez l'utiliser

Q5 - Quand j'ajoute un accent dans le commentaires, le programme indique une erreur ?

En principe, cette erreur ne devrait pas se produire en python3.6 mais uniquement en python2.7. Pour éviter ce problème d'accent, il faut juste écrire à la toute première ligne du programme python l'instruction suivante :

```
# -*- coding: utf-8
```

Q6 - Comment lire un fichier de données en Python avec numpy (méthode la plus simple) ?

Au lieu de lire le tableau en python et de le convertir en numpy comme cela a été présenté à la faq Q4, on peut aussi directement lire le tableau en numpy

```
nom_fich = 'data.txt'  # definition du nom du fichier
mesures = np.loadtxt(nom_fich) # lecture du fichier dans le tableau mesures
```

Q7 - Comment lire un fichier de données en Python sans utiliser numpy (méthode plus compliquée) ?

On considère un fichier data.txt contenant un tableau de mesures sous forme des lignes et de colonnes. Le fichier data.txt peut, par exemple, contenir 4 séries de 3 mesures (4 lignes, 3 colonnes)

```
25.1  30.0  34.9
25.0  29.9  35.0
24.8  30.2  35.1
25.2  30.1  35.0
```

L'exemple suivant montre comment remplir un tableau python ou numpy avec les données du fichier data.txt

```
nom_fich = 'data.txt'  # definition du nom du fichier
fich = open(nom_fich,"r") # ouverture du fichier en lecture ("r" est facultatif)
mesures_fich = [] # création d'un tableau vide pour recevoir les mesures
for l in fich.readlines() :  # lecture d'une ligne du fichier dans l
    l = l[0:-1] # enleve le dernier caractère (retour à la ligne)
    ls = l.split(" ") # decoupe la ligne en détectant les espaces comme
                      #  séparation
    v=[] # création d'un vecteur vide
    nc = len(ls) # nombre de valeurs trouvées dans la ligne (nombre de colonnes)
    for i in range(nc):
        try:
            v.append(float(ls[i])) # ajoute les valeurs au vecteur v
        except:
            pass  # ne fait rien si ce n'est pas une valeur flottante
    mesures_fich.append(v) # ajoute le vecteur au tableau de mesures
fich.close() # ferme le fichier car la lecture est terminée
```

mesures_fich est un tableau python, on peut le convertir en tableau numpy de la façon suivante :

```
# transforme le tableau de mesure en un tableau numpy
mesures = np.asarray(mesures_fich)
```

Q8 - Comment réaliser des opérations de calcul matriciel de base avec Python et numpy ?

En début de programme, on ajoute la ligne suivante :

```
import numpy as np
```

np c'est juste pour taper moins de caractères dans la suite du programme car pour accéder à une fonction de numpy, on fera np. au lieu de numpy. Cette convention est généralement adoptée par la communauté des utilisateurs Python.
Les principales fonctions utiles sont : np.zeros(), np.ones(), np.linspace(), .dot(), np.power()

```
n = 10
z = np.zeros(n)
o = np.ones(n)
p = np.linspace(1,n,n) # n valeurs de 1 à n
x = o*2
y = np.power(x,p) # power of 2
print (z)
print (o)
print (p)
print (x)
print (y)
```

Le résultat est :

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
[  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
[ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
[    2.     4.     8.    16.    32.    64.   128.   256.   512.  1024.]
```

Q9 - Comment calculer la moyenne des colonnes d'un tableau avec Python et numpy ?

Pour calculer la moyenne d'un vecteur, il faut, sans calcul matriciel, utiliser un boucle de calcul

```
v=[1.,2.,3.,2.,1.] # définition de v
n=len(v) # nombre d'élements (de composantes) de v
s=0.0 # initialize la somme
for i in range(n):
    s += v[i]  # somme tous les éléments du vecteur
m = s/float(n) # calcule la moyenne
```

autre méthode plus "pythonesque"

```
v=[1.,2.,3.,2.,1.] # définition de v
n=len(v) # nombre d'élements (de composantes) de v
s=0.0 # initialise la somme
for x in v:
    s += x  # somme tous les éléments du vecteur
m = s/float(n) # calcule la moyenne
```

Avec numpy, le module de calcul matriciel de Python, cela devient plus simple. Pour la moyenne d'un vecteur on procède comme suit :

```
v=np.array([1.,2.,3.,2.,1.]) # définition de v sous forme de tableau numpy
m=np.mean(v) # calcul de la moyenne
```

Différentes moyennes (ligne, colonne, globale) peuvent être calculées sur une matrice :

```
# definition de la matrice
a = np.asarray([[1.,2.,-1.,0.],[0.,0.,1.,-1.],[2.,1.,-3.,-2.]])
print (a) # affiche la matrice
print (a.shape)  #  matrice de 3 lignes de 4 colonnes
print (np.mean(a)) # moyenne sur toute la matrice (scalaire)
print (np.mean(a,0)) # vecteur avec la moyenne de chaque colonne
print (np.mean(a,1)) # vecteur avec la moyenne de chaque ligne
```

Les résultats des commandes ci dessous sont les suivants :

```
[[ 1.  2. -1.  0.]
 [ 0.  0.  1. -1.]
 [ 2.  1. -3. -2.]]
(3, 4)
0.0
[ 1.  1. -1. -1.]
[ 0.5  0.  -0.5]
```

...

Q10 - Comment calculer l'écart-type d'un vecteur avec numpy ?

Soit **v** un vecteur de **n** mesures, la fonction numpy **np.std(v,ddof=1)** permet d'estimer l'écart-type de la série de mesure **v**. Attention, si **n** est le nombre de mesures (nombre d'éléments de **v**), la fonction **np.std(v)** divise par **n**. Pour calculer l'écart type ("non biaisé"), il faut diviser par **n-1** et c'est pourquoi l'option **ddof=1** est ajoutée.
exemple :

```
n = 10
v = np.random.normal(loc=0.0,scale=0.1,size=n)
std_bias = np.std(v)
std_ok = np.std(v,ddof=1)
print ("ecart-type biaisé = %.2f"%(std_bias))
print ("ecart-type = %.2f"%(std_ok))
```

le résultat est :

```
ecart-type biaisé = 0.04
ecart-type = 0.05
```

Q11 - Comment calculer l'écart-type des colonnes d'un tableau avec Python et numpy ?

Création artificielle d'un tableau de mesures bidon ym

```
xm = np.linspace(1,10,10) # mesurande
ym = np.zeros((5,10)) # matrice 2D de mesures
for i in range(len(xm)):
   ym[:,i] = xm[i] +np.random.normal(loc=0.0,scale=0.1,size=5) # 5 mesures bruitées
print (ym)
```

Le calcul de l'écart-type pour chaque valeur de mesurande (chaque colonne) se fait avec la fonction np.std()

```
sig = np.std(ym, 0, ddof=1)
print (sig)  # ecart type par colonne
```

Q12 - Comment calculer la valeur maximale d'un vecteur avec Python et numpy ?

Création artificielle d'un tableau de mesures bidon ym

```
xm = np.linspace(1,10,10) # mesurande
ym = 10.0-(xm-5)**2
print (ym)
```

La valeur maximale est donnée par la fonction np.max() et sa position (index) dans le vecteur est donnée par np.argmax()

```
ymx = np.max(ym)
imx = np.argmax(ym)
print ("max=",ymx,"à la position",imx)
```

Le résultat est :

```
[ -6.   1.   6.   9.  10.   9.   6.   1.  -6. -15.]
max= 10.0 à la position 4
```

Q13 - Comment estimer la meilleure droite passant par les points de mesure ?

Nous utilisons pour cela la fonction **np.polyfit** qui permet de trouver le polynôme qui correspond le mieux aux données. Pour une droite, le degré **deg** est égal à 1.

```
xm = np.linspace(1,10,10) # mesurande
a_theo = 2.0
b_theo = -1.0
ym = a_theo*xm + b_theo + np.random.uniform(-0.2,0.2,10)  # mesures bruitées
deg = 1 # degré du polynome
a_lin, b_lin = np.polyfit(xm, ym, deg) # pente , ordo. à l'orig.
print ("pente = %.2f (théorique = %.2f)"%(a_lin,a_theo))
print ("ord. orig. = %.2f (théorique = %.2f)"%(b_lin,b_theo))
ylin = a_lin*xm + b_lin # meilleure droite
```

Le résultat est :

```
pente = 1.98 (théorique = 2.00)
ord. orig. = -0.95 (théorique = -1.00)
```

Q14 - Comment afficher des courbes avec Python ?

On reprend l'exemple de la question précédente, et on souhaite afficher les deux courbes :

```
plt.figure()
plt.plot(xm,ym)
#plt.hold(True)   # utile dans certains cas
plt.plot(xm,ylin,'r')
```

Q15 - Comment fixer le nombre de décimales lors de l'affichage d'un vecteur numpy ?

Pour fixer à n le nombre de décimales d'une valeur réelle v, il faut utiliser l'arrondi à n décimales : **np.round(v,n)**. Pour afficher les n décimales il faut utiliser la fonction **np.set_printoptions()** comme indiqué dans l'exemple ci-dessous ou l'on conserve 2 décimales.

```
import numpy as np
v = [3.127,-1.989,-0.004]
print (v)
v2d = np.round(v,2)
print (v2d)
np.set_printoptions(precision=2,formatter={'float': '{: .2f}'.format})
print (v2d)
```

Le résultat est :

```
[3.127, -1.989, -0.004]
[ 3.13 -1.99 -0.  ]
[ 3.13 -1.99 -0.00]
```

Remarque : pour les très faibles valeurs négatives, le 0 est précédé du signe -.

Q16 - Comment fixer le nombre de décimales lors de l'affichage d'un scalaire (non numpy) ?

En Python, il est possible de contrôler le format du texte affiché à la console. Voici quelques exemples :

```
a = -1.9899
ia = round(a)
# entier le plus proche : format standard
print ("arrondi : ",ia)
# entier le plus proche : format entier %d
print ("arrondi : %d"%(ia))
# entier le plus proche : format entier 8 caractères %8d
```

```
print ("arrondi : %8d"%(ia))
# entier le plus proche : format entier 8 caractères avec 0 %8.8d
print ("arrondi : %8.8d"%(ia))
fa = round(a,3)
# valeur réelle
print ("a =",a)
# valeur à trois décimales (nomb
print ("a =",fa)
# valeur à trois décimales (avec 3 digits)
print ("a = %.3f"%(fa))
```

Le résultat est :

```
arrondi :  -2
arrondi : -2
arrondi :       -2
arrondi : -00000002
a = -1.9899
a = -1.99
a = -1.990
```

Q17 - Comment manipuler l'hexadécimal en Python ?

Un processeur représente toujours les données numériques sous forme binaire, ce qui est assez difficile à interpréter par l'utilisateur. C'est pourquoi la base d'affichage par défaut est la base 10, mieux maîtrisée par les utilisateurs. Dans certains cas, il peut être intéressant d'utiliser la base hexadécimale.
En Python, il est possible de contrôle le format du texte affiché à la console. Le code suivant montre comment passer du décimale à l'hexadécimal et vice-versa.

```
a = 14 # entrée d'une valeur sous forme décimale
print ("décimal     :",a)  # affichage décimal standard
print ("hexadécimal : %x"%(a)) # affichage en hexadécimal
print ("hexadécimal : %2.2x"%(a)) # affichage en hexadécimal (2 digits)
a = 0xff # entrée d'une valeur sous forme hexadécimale
print ("décimal     :",a)  # affichage décimal standard
print ("hexadécimal : %x"%(a)) # affichage en hexadécimal
print ("hexadécimal : %2.2x"%(a)) # affichage en hexadécimal (2 digits)
```

Le résultat est :

```
décimal     : 14
hexadécimal : e
hexadécimal : 0e
décimal     : 255
hexadécimal : ff
hexadécimal : ff
```