

UED SIM – SysIn

Séance TD : Programmation procédurale - Exemples en langage *Python*[™]

Jean-luc Charles (jean-luc.charles@ensam.eu)

Éric Ducasse (eric.ducasse@ensam.eu)

Préambule

- Créer un répertoire **TD-TraceCourbe** dans votre espace de travail
- Copier dans ce répertoire les documents téléchargés depuis la plateforme E-Learning SAVOIR "BO-Systèmes informatiques : Le langage Python/TD Lecture fichiers ASCII et tracés de courbes"
- **Tout le travail qui suit se fera dans votre répertoire TD-TraceCourbe.**

Objectifs de la séance

Les objectifs de cette séance sont :

- ✓ Mettre en oeuvre la programmation procédurale sur des exemples simples.
- ✓ Savoir résoudre un problème en le découpant en tâches élémentaires correspondant à des **fonctions**.
- ✓ Comprendre la structure d'un fichier ASCII de données.
- ✓ Savoir lire un fichier ASCII, ligne par ligne.
- ✓ Savoir tracer des courbes, avec des données lues dans un fichier ASCII.
- ✓ Connaître les principaux formats de fichiers images bitmap et leurs principales caractéristiques (bitmap/vectorel, compression, transparence...).
- ✓ Savoir générer un fichier image *bitmap* représentant le tracé de courbes.

Les exemples de programmation procédurale en langage Python seront centrés sur les points suivants :

- lire des données dans un fichier ASCII délimité,
- tracer des courbes,
- enregistrer les tracés dans un fichier image,
- naviguer dans une arborescence.

Les images au format matriciel (*bitmap*)

Une image matricielle (*bitmap*, en mode point...) affichée est constituée d'un ensemble ordonné de pixels (points de couleur) qui constituent l'image à l'écran (ou l'image imprimée). Chaque pixel possède plus ou moins d'informations : couleur (codage RVB, CMY...), intensité, transparence...



Pour une taille fixée en centimètres, plus une image possède de pixels, plus les détails sont fins, plus la taille de l'image en octets est grande.

=> on parle de la **résolution** d'une image :

- en *ppi* (*pixel per inch*) pour une image à l'écran,
- en *dpi* (*dot per inch*) pour une image imprimée.

Les formats de fichiers image numérique

([wikipédia : Images Numériques](#))

<i>Format</i>	<i>type</i>	<i>compression</i>	<i>perte</i>	<i>codage couleur</i>	<i>anim.</i>	<i>transp.</i>
JPEG	bitmap	oui	avec	24 bits (16 millions)	non	non
GIF	bitmap	oui	sans	8 bits (256)	oui	oui
PNG	bitmap	oui	sans	24 bits (16 millions)	non	oui
TIFF	bitmap	oui/non	avec/sans	24 bits (16 millions)	oui	oui
SVG	vectorel	possible		24 bits (16 millions)	oui	oui

Présentation du problème

Une machine de traction fournit les résultats d'un essai dans un fichier **ASCII délimité** contenant les colonnes suivantes :

- 1^{re} colonne : instant d'acquisition (secondes)
- 2^{me} colonne : contrainte mesurée (Pa)
- 3^{me} colonne : déformation mesurée (pourcentage de la longueur initiale).

☞ Noter les lignes de commentaires, commençant par le caractère '#'

Time [s]	Stress [Pa]	Strain
0.000000e+00	3.321566e+05	4.340203e-05
1.700000e-01	6.013489e+06	3.299359e-04
3.400000e-01	1.154852e+07	9.718301e-04
5.100000e-01	1.710066e+07	1.328353e-03
6.800000e-01	2.265946e+07	1.854428e-03
8.500000e-01	2.875566e+07	2.224948e-03
1.020000e+00	3.154837e+07	2.622302e-03
1.190000e+00	3.262366e+07	3.047014e-03
1.360000e+00	3.175871e+07	3.466145e-03
1.530000e+00	3.336673e+07	4.022022e-03
1.700000e+00	3.271517e+07	4.273263e-03
1.870000e+00	3.275063e+07	4.913872e-03
2.040000e+00	3.448111e+07	5.250066e-03
2.210000e+00	3.460045e+07	5.666204e-03

Creation : Jeudi 3 novembre 2011, 23:09:24

Cahier des charges

À partir du fichier *data001.txt*, générer le fichier image *data001.png* contenant les 2 graphes :

- contrainte et déformation en fonction du temps (graphe à 2 échelles verticales différentes)
 Axe horizontal : Temps, $t_{min} = 0$, $t_{max} = 2.3$ (en secondes), label "Temps [s]"
 Axe vertical gauche : Contrainte, $y_{min} = 0$, $y_{max} = 50$ (en MPa), label "Contrainte [MPa]"
 Axe vertical droit : Déformation, $y_{min} = 0$, $y_{max} = 10^{-2}$, label "Déformation"
- graphe de la contrainte fonction de la déformation
 axe horizontal (déformation) : $y_{min} = 0$, $y_{max} = 6 \times 10^{-3}$, label "Déformation"
 axe vertical (contrainte) : $y_{min} = 0$, $y_{max} = 40$ (en MPa), label "Contrainte [MPa]"

Le travail sera éventuellement fait avec un tableur (cf instructions enseignant), puis à l'aide d'un programme *Python*[™].

Utilisation d'un tableur (ne pas consacrer plus de 30 minutes à cette étape!)

- Ouvrir le fichier *data001.txt* avec un tableur
- Utiliser l'assistant d'importation du tableur pour obtenir une importation correcte du fichier ASCII délimité...
- Attention au séparateur décimal utilisé dans le fichier (point), qui peut être différent de celui utilisé par défaut par le tableur (virgule)
- Créer les 2 graphes selon les spécifications demandées (cf. diapo précédente "Cahier des charges")
- Trouver un moyen de créer une image au format PNG contenant les 2 graphes l'un en dessous de l'autre
- Ouf!

Python : Lecture des lignes d'un fichier ASCII

- Avec **IDLE**, ouvrir le fichier Python *LireFichier.py*
- Faire interpréter ce fichier dans le shell, et observer le résultat...
- Examiner le contenu de l'objet `lu` dans le shell Python...
- Régler les problèmes éventuels...
- Modifier le programme pour ne pas ajouter les lignes de commentaire à la liste `lu`

```
File Edit Format Run Options Windows Help
dataFile="data001.txt"
#
# Lecture du fichier ligne par ligne
#
lu=[]
f = open(dataFile, 'r')
for line in f:
    print line
    lu.append(line)
f.close()
```

Constitution des 3 listes time, stress et strain

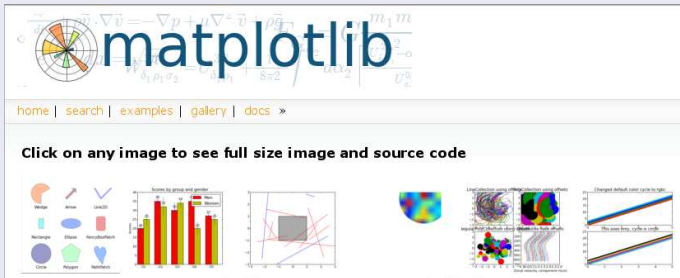
Expérimenter les commandes qui suivent dans le shell Python :

```
>>> lu[5]
>>> lu[5].split()
>>> map(float, lu[5].split())
>>> t,c,d=map(float, lu[5].split())
>>> t
>>> c
>>> d
```

- En déduire l'évolution du programme permettant de construire les 3 listes time, stress et strain à l'aide d'une boucle balayant la liste `lu`...
☞ on pourra déclarer les 3 listes avec : `time, stress, strain = [], [], []`
- Exécuter le programme et vérifier le contenu des listes time, stress et strain
- Modifier le programme pour avoir la liste stress en MPa.

Tracé des courbes I : module Python **matplotlib**

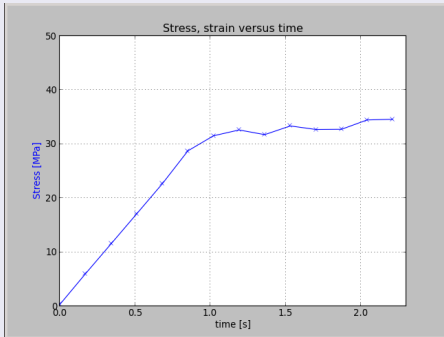
- Le module Python **matplotlib** fournit des extensions très puissantes pour effectuer des tracés sous Python "à la MatLab"
- Pour avoir une idée des possibilités offertes par **matploblib**, visiter la galerie <http://matplotlib.org/gallery.html>



- Pour la prise en main rapide de matplotlib, on peut consulter les premières pages du tutorial du module **matplotlib.pyplot** http://matplotlib.org/users/pyplot_tutorial.html

Tracé des courbes II : contrainte en fonction du temps

- rajouter l'import du module **matplotlib.pyplot** sous le nom **plt** en début de votre programme :
`import matplotlib.pyplot as plt`
- Compléter votre programme Python avec les lignes indiquées ci-contre... à détailler
- Faire interpréter ce programme dans le shell...



```
File Edit Format Run Options Windows Help

#
# Graphe des courbes
#

# Graphe stress versus time
plt.figure()
plt.title("Stress, strain versus time")
plt.grid(True)
plt.xlabel('time [s]')
plt.xlim(0.,2.3)
plt.ylabel('Stress [MPa]',color='b')
plt.ylim(0.,50)
plt.plot(time,stress,'x-b')

plt.show()

Ln: 29 Col: 15
```

Tracé des courbes II : contrainte et déformation en fonction du temps

- Compléter votre programme Python avec les lignes indiquées ci-dessous... à détailler
- Faire interpréter ce programme dans le shell, et observer le résultat...

```
File Edit Format Run Options Windows Help

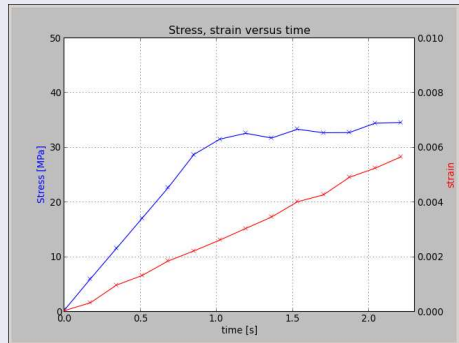
#
# Graphe des courbes
#

# Graphe stress, strain versus time
plt.figure()
plt.title("Stress, strain versus time")
plt.grid(True)
plt.xlabel('time [s]')
plt.xlim(0.,2.3)
plt.ylabel('Stress [MPa]',color='b')
plt.ylim(0.,50)
plt.plot(time,stress,'x-b')

plt.twinx()
plt.ylim(0.,10.e-3)
plt.xlabel(0.,2.3)
plt.ylabel('strain',color='r')
plt.plot(time,strain,'x-r')

plt.show()

Ln: 44 Col: 0
```



Tracé des courbes III : les 2 graphes

- Rechercher la documentation sur la fonction `matplotlib.pyplot.subplot` (tracé de sous figures) avec le **Quick search** de la page matplotlib <http://matplotlib.org>
- Compléter votre programme Python avec les lignes ci-dessous... à détailler, exécuter...

```
File Edit Format Run Options Windows Help
# Graphe des courbes
#

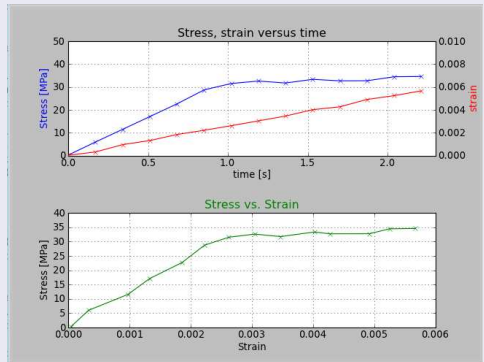
# Graphe stress, strain versus time
plt.figure()
plt.subplot(211)      # 2 lignes, 1 colonne, graphe 1
plt.subplots_adjust(hspace=0.5)
plt.title("Stress, strain versus time")
plt.grid(True)
plt.xlabel('time [s]')
plt.xlim(0.,2.3)
plt.ylabel('Stress [MPa]',color='b')
plt.ylim(0.,50)
plt.plot(time,stress,'x-b')

plt.twinx()
plt.xlim(0.,2.3)
plt.ylim(0.,10.e-3)
plt.ylabel('strain',color='r')
plt.plot(time,strain,'x-r')

# Graphe stress versus strain
plt.subplot(212)      # 2 lignes, 1 colonne, graphe 2
plt.title("Stress vs. Strain", color='g')
plt.grid(True)
plt.xlabel('Strain')
plt.xlim(0.,6.e-3)
plt.ylabel('Stress [MPa]')
plt.ylim(0.,40)
plt.plot(strain,stress,'x-g')

plt.show()
```

Ln: 54 Col: 29



Export des tracés en fichiers bitmap

- Dans la page de recherche **matplotlib** (<http://matplotlib.org>) trouver des informations sur la fonction `matplotlib.pyplot.savefig`
- En utilisant la méthode *adhoc* de la classe **str**, définir l'objet `imageFile` qui contient le nom du fichier ASCII dans lequel on a remplacé l'extension `".txt"` par `".png"`
Cet objet sera passé en argument à la fonction `savefig...`
- En déduire les lignes de code à insérer avant l'instruction `plt.show()` pour faire générer un fichier image *adhoc*, faire exécuter et vérifier la présence et le contenu du fichier image...

Critique du travail effectué, méthodologies proposées

- Critique du programme développé dans le fichier *LireFichier.py* :
 - ☞ résout le problème ponctuel posé, mais ...
 - fichier "fourre-tout", qui mélange beaucoup de choses sans proposer d'outils élémentaires RÉ-UTILISABLES....
 - couteux à modifier pour l'adapter à un autre problème...
 - à jeter après usage !
 - ...
- **Méthodologie de développement**
 - 1) découpage des tâches en fonctions élémentaires :
 - ☞ création d'un fichier (module Python) *TraiterTCD.py* contenant des **fonctions paramétrables** (arguments), **maintenables**, **réutilisables**...
 - 2) introduction d'une section de test unitaire des fonctions du module **TraiterTCD**

```
if __name__=='__main__' :  
    ... exécution des fonctions avec des données test...
```
- **Méthodologie d'organisation de la structure des fichiers**
 - l'arborescence (répertoire/sous-répertoire/...) **doit** être utilisée pour structurer l'espace de travail...
 - il **faut** être capable de naviguer dans l'arborescence avec le programme de traitement des données !

Approche procédurale : écriture du module **TraiterTCD**

- ① sauvegarder le fichier *LireFichier.py* sous le nom *TraiterTCD.py*
- ② modifier le code du fichier *TraiterTCD.py* pour faire apparaître les 2 fonctions LireFichierTCD et TracerTCD :
 - ajouter les arguments permettant de paramétrer le travail des deux fonctions :

LireFichierTCD	- le nom du fichier à lire
TracerTCD	- les trois listes temps, contrainte et déformation - un paramètre nommé name (valeur par défaut : None) servant de switch entre le tracé des courbes "à l'écran / dans un fichier Image" : <pre>if name == None: # tracé écran else: # tracé dans le fichier nommé name</pre>

- le cas échéant, prendre soin de faire retourner les grandeurs utiles (instruction Python return)
- ③ introduire à la fin du fichier un bloc de tests commençant par la ligne :

```
if __name__ == '__main__':
```
 - ④ écrire quelques lignes dans ce bloc de test permettant de tester le fonctionnement des 2 fonctions définies dans le module

Balayer une arborescence : module Python `os`

Dans un shell, essayer les commandes suivantes :

```
>>> import os
>>> os.getcwd()
>>> os.listdir(".")
>>> os.mkdir("tmp")
>>> os.listdir(".")
>>> os.listdir("ResultatsTraction")
>>> os.listdir("ResultatsTraction/Essais")
>>> "data19.txt" in os.listdir("ResultatsTraction/Essais")
>>> "ivgizgzi" in os.listdir("ResultatsTraction/Essais")
>>> "ivgizgzi" not in os.listdir("ResultatsTraction/Essais")
```

Programme *TraiterRepertoire.py* : étape 1

Construire le programme *TraiterRepertoire.py* :

- ❶ qui importe les 2 fonctions `LireFichierTCD` et `TracerTCD` du module **TraiterTCD**,
- ❷ qui définit la fonction `TraiterRepertoire`, prenant en argument `repName` (nom du répertoire à traiter)
- ❸ qui vérifie la présence du répertoire *Essais* dans le répertoire à traiter, et en cas de succès :
 - qui construit la liste `Ld` des noms des fichiers du répertoire *Essais*
 - qui vérifie que la liste `Ld` n'est pas vide
 - qui teste la présence du répertoire *Courbes* sous le répertoire de travail, et le crée si besoin.

Introduire à la fin du fichier un bloc test commençant par la ligne :

```
if __name__ == '__main__':
```

écrire quelques lignes dans ce bloc pour exécuter le programme et vérifier son fonctionnement.

Programme *TraiterRepertoire.py* : étape 2

Faire évoluer le programme *TraiterRepertoire.py* pour réaliser les traitements de tous les fichiers *dataXY.txt* du répertoire *Essais* :

- ① lecture du fichier *dataXY.txt*
- ② exportation du tracé des courbes dans un fichier image *dataXY.png* au format PNG, dans le répertoire *Courbes*

Tester...

Programme *TraiterRepertoire.py* : étape 3

Modifier la fonction *TraiterRepertoire* du module **TraiterRepertoire** pour qu'elle prenne un deuxième argument :

- nommé *ecran*, ayant **False** pour valeur par défaut,
- permettant d'implémenter le comportement :

```
if ecran == True:
    # courbes tracées à l'écran
else:
    # courbes dans les fichiers PNG
```

Tester...

Conclusion

Peut-on faire facilement ce travail avec un tableur ☺ ?

Bonus - Programme *TraiterRepertoire.py* : étape 4

Pour chacun des essais traités, on doit calculer l'énergie volumique de déformation, et écrire le résultat dans de nouveaux fichiers (ASCII tabulé) *dataXY-W.txt* contenant les quatre champs : temps, contrainte, déformation et énergie volumique de déformation.

Description du format des fichiers :

- La première ligne est un commentaire qui donne les unités des différents champs :
"# time[s] stress[MPa] strain[%] energyDensity[MJ.m⁻³]"
- Les lignes suivantes donnent les valeurs des quatre champs, au format flottant scientifique avec 4 chiffres après la virgule, séparés par le caractère ASCII TAB.

En suivant la méthodologie liée à l'approche procédurale (diapo 14), proposer une évolution du module **TraiterTCD** pour traiter ce problème.

Modifier le programme *TraiterRepertoire.py* pour faire écrire les fichiers *dataXY-W.txt* dans le répertoire *DataW*.

Indications : Pour l'écriture formatée des données dans le fichier, essayer dans le shell :

```
>>> a,b=100,0.2
>>> "%.4e\t%.4e" % (a,b)    # que représente le caractère '\t' ?
>>> print "%.4e\t%.4e" % (a,b)
>>> print "%.4e\t%.4e\n" % (a,b) # que représente le caractère '\n' ?
```

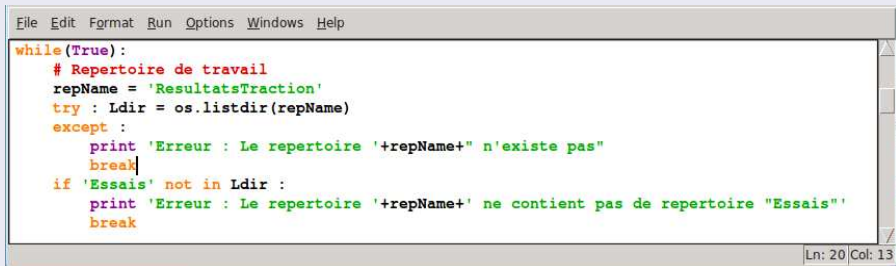
Bonus - Programme *TraiterRepertoire.py* : étape 5

Création d'un document PDF de synthèse contenant tous les tracés :

- Visiter le *How to* de matplotlib sur la question "*Save multiple plots to one pdf file*" (http://matplotlib.org/faq/howto_faq.html)
- mettre en oeuvre ...

Bonus - Programme *TraiterRepertoire.py* : étape 6

Gestion des erreurs : en vous inspirant de l'exemple donné, implémenter la gestion des erreurs dans le programme *TraiterRepertoire.py*.



```
File Edit Format Run Options Windows Help
while(True):
    # Repertoire de travail
    repName = 'ResultatsTraction'
    try : Ldir = os.listdir(repName)
    except :
        print 'Erreur : Le repertoire '+repName+' n'existe pas'
        break
    if 'Essais' not in Ldir :
        print 'Erreur : Le repertoire '+repName+' ne contient pas de repertoire "Essais"'
        break
```

Ln: 20 Col: 13