TP7 Utilisation de modules, de bibliothèques

PCSI$_2$ L. Parise – N. Wipf – S. Limal – D. Mengel, 2025 – 2026

Objectifs de la séance : savoir réaliser les opérations suivantes :

- Lecture d'un fichier de données simples.
- Calculs statistiques sur ces données.
- Représentations graphiques (courbes, histogrammes).

I Manipulation d'un fichier de données

Lors d'une séance de TP de sciences expérimentales, il est courant d'obtenir une grande quantité de données à traiter. Il apparaît judicieux de stocker ces données sous la forme d'un fichier texte indépendant dont le contenu pourra être lu et exploité par la suite. On distingue ainsi :

- le fichier texte (format .txt ou .csv) contenant les données et
- le script en langage Python (format .py) qui a pour but de les exploiter.

I.1 Création d'un fichier

En python on utilise la fonction `open` avec en paramètres

- le nom du fichier à créer et
- le mode d'ouverture `'w'` pour le mode écriture (*write*).

Le code suivant permet de créer le fichier `nom_du_fichier` et d'y écrire la chaine de caractère `"Chaine"`.

```
fic = open("nom_du_fichier", 'w')
fic.write("Chaine")
fic.close()
```

Remarques :

- D'autres options sont possibles, en particulier l'encodage du fichier `encoding="latin1"` pour un fichier encodé par Windows ou `encoding="utf8"` pour un fichier encodé par le reste du monde.
- Une fois que le fichier est créé, l'interpréteur Python maintient un marqueur (fictif) à la position courante, indiquant où seront écrits les prochains caractères.
  Toute opération d'écriture fait ensuite bouger ce pointeur vers l'avant : `fic.write("Chaine")` permet d'écrire la chaine de caractères `"Chaine"` à la position courante du pointeur et déplace ce dernier derrière le dernier caractère ajouté.
- Pour écrire sur plusieurs lignes dans le fichier il faut ajouter `\n` à chaque fin de ligne.
- Si le mode d'ouverture est `'w'` et que le fichier `"nom_du_fichier"` n'existe pas encore il sera créé. Par contre, si ce fichier existe déjà, il sera écrasé (vidé avant utilisation) donc **attention au risque de pertes de données !** Il existe également le mode `'a'` (*append*) qui place le marqueur à la fin d'un fichier existant puis ajoute `"Chaine"` à chaque utilisation de `fic.write("Chaine")`.

Question 1.

1. Ouvrez maintenant Pyzo et créez un nouveau fichier nommé `TP07_Modules.py` dans un répertoire de votre choix.
2. En complétant le code suivant dans l'éditeur, créez un fichier `Mon_premier_fichier.txt` contenant les chaines de caractères suivantes sur trois lignes :
   - Première ligne.
   - 12 ; 4 ; 8
   - Fin du fichier.

```
fic=open('Mon_premier_fichier.txt','w')
fic.write(?)
fic.write('12 ; 4 ; 8\n')
fic.write('Fin du fichier.')
fic.close()
```

3. Il est probable que le fichier `Mon_premier_fichier.txt` ne soit pas créé dans le même répertoire que `TP07_Modules.py` (sauf si vous avez coché **"Change directory when executing file"** dans le menu **"Run"**).
   Vous devez pour cela exécuter le fichier `TP07_Modules.py` à l'aide de la commande **"Exécuter en tant que script"** (au moins lors de la première exécution, les exécutions suivantes pouvant être classiques) : menu **"Run"** puis **"Run file as script"** de **Pyzo** ou raccourcis Ctrl-Shift-E sous Windows ou command-Shift-E sous MacOs.
4. Vérifiez ensuite que tout s'est bien passé en ouvrant directement le fichier `Mon_premier_fichier.txt` avec l'application Bloc-Notes de Windows ou textEdit de MacOs par exemple.

I.2 Ouverture d'un fichier en mode lecture

On utilise là aussi la fonction open mais cette fois avec le mode d'ouverture `'r'` (*read*).

```
fic = open("nom_du_fichier", 'r')
# suite d'instructions :  fic.read(), fic.readline() ...
fic.close()
```

Attention, ici aussi et comme nous l'avons déjà vu lors de TP précédents, si les fichiers .txt et .py sont enregistrés dans des dossiers différents, une erreur du type

FileNotFoundError : [Errno 2] No such file or directory : 'nom_du_fichier'

sera affichée lors de l'exécution du fichier `TP07_Modules.py`. L'interpréteur Python indique qu'il ne connaît aucun fichier texte à ce nom. Par contre, si ils sont bien enregistrés dans un même dossier, l'importation du fichier texte ne posera plus de problème en exécutant le fichier en tant que script : **"Run file as script"**.

Comme tout à l'heure, une fois le fichier .txt ouvert, l'interpréteur Python maintient un marqueur à la position courante qui évolue au fur et à mesure de l'exécution des instructions.

Question 2.

1. Toujours dans l'éditeur, complétez le fichier `TP07_Modules.py` avec le code donné ci-dessous.

```
fic = open("Mon_premier_fichier.txt", 'r')
#à partir de la position du marqueur
L=fic.readlines() #retourne la liste de toutes les lignes
#L=fic.read() #retourne toutes les lignes
#L=fic.readline() #retourne les caractères de la ligne
#L=fic.readline(6) #retourne les 6 caractères suivants
fic.close() # Fermeture du fichier

print(L)
```

Décommentez successivement une seule ligne à la fois puis exécutez afin de comprendre (ou confirmer) à quoi correspond chaque instruction.

2. Pour récupérer les données numériques de la seconde ligne (si c'est cela qui nous intéresse) on peut utiliser la commande `ligne.split(";")` sur cette ligne.

```
fic=open('Mon_premier_fichier.txt','r')
L=fic.readlines() #retourne la liste de toutes les lignes
fic.close() # Fermeture du fichier
Ligne_2=L[?] #on récupère la ligne qui nous intéresse
x,y,z=Ligne_2.split(?) #on sépare les données de la ligne
print(float(x),float(y),float(z)) #affichage des données numériques
```

Remarque : pour une ligne où les données sont séparées par des tabulations, il faudra utiliser `ligne.split("\t")`

II Utilisation de modules (bibliothèques)

II.1 Importer et utiliser un module

Définition : on appelle "module" tout fichier constitué de code Python (c'est-à-dire tout fichier avec l'extension . py) importé dans un autre fichier ou script.

Les modules permettent la séparation et donc une meilleure organisation du code. Ils contiennent en effet des fonctions qu'on peut ensuite appeler depuis le programme principal.

Exemple : exécuter le code suivant `cos(1)` dans la console renvoie l'erreur

NameError : name 'cos' is not defined

Pour utiliser cette fonction il faut au préalable importer un module qui la contient. Prenons par exemple le module `numpy` très utile pour les calculs numériques et qui contient de nombreuses fonctions mathématiques.

L'appel de la fonction dépendra ensuite de la manière dont on a effectué l'importation.

Exemples :

	Commande	Appel de la fonction
Importation du module	`import numpy`	`numpy.cos()`
	`import numpy as np`	`np.cos()`
Importation des fonctions utiles seules	`from numpy import cos`	`cos()`

Remarque : `np` est un alias que l'on peut choisir librement. Cette méthode d'importation (que l'on privilégiera) est particulièrement utile lorsque le nom d'un module est long.

Question 3.

1. En entête de votre programme `TP07_Modules.py`, placez un paragraphe `## Bibliotheques` dans lequel vous importerez le module `numpy` avec l'alias `np` et le module `matplotlib.pyplot` avec l'alias `plt`. Ce module est utilisé pour effectuer le tracé de graphes.
2. Effectuez le tracé de la fonction $\cos(x)$ pour $0 \leq x \leq 4\pi$.

```
x=np.linspace(0,4*np.pi,200) #tableau des valeurs de x, il peut aussi s'
    agir d'une liste
y=np.cos(x) #calcul des valeurs de cos(x) : tableau ou liste de même
    taille que x
```

```
plt.figure("Mon graphe") #évite de superposer des tracés sur un même
    graphe
plt.plot(x,y,label="Courbe sinusoïdale") #le tracé en lui-même
plt.title("Fonction cosinus") #facultatif et tellement indispensable à
    la fois !
plt.xlabel('$x$')
plt.ylabel('$\cos(x)$')
plt.show() #affichage de la fenêtre. Ligne à commenter (#) pour éviter l
    'affichage à chaque exécution
```

Remarques :

- La fonction `dir(np)` permet de lister le contenu du module d'alias `np`. Testez la dans la console.
- Pour obtenir des informations sur une fonction particulière vous pouvez utiliser la commande `help()` dans la console. Par exemple `help(np.cos)`. En cas d'erreur, essayez `np.info(np.cos)`. Vous pourrez à cette occasion remarquer la différence entre la fonction `cos` du module math et celle du module `numpy`.

II.2 Création d'un module

Bonne nouvelle, vous pouvez créer votre propre bibliothèque de fonctions !

Question 4. construisons maintenant une bibliothèque `biblio_statistiques.py` qui ne devra contenir que des définitions de fonctions. Ce module sera importé dans le paragraph `## Bibliothèques` de votre fichier `TP07_Modules.py`

Comme dans le cas du fichier texte, les deux fichiers doivent se situer dans le même dossier.

1. Depuis Pyzo, créez un nouveau fichier et enregistrez le immédiatement sous le nom `biblio_statistiques.py` dans le même répertoire que `TP07_Modules.py`
2. Dans le fichier `biblio_statistiques.py`, définir la fonction `moyenne(L)` qui renvoie la valeur moyenne $\overline{x}$ d'une liste `L` contenant N éléments x_i avec $i = 0, \ldots, N-1$

```
import numpy as np

def moyenne(liste):
    "Renvoie la moyenne des valeurs de la liste" #documentation de la
    fonction
    N=? #nombre de valeurs dans la liste
    somme=? #initialisation
    for ...
        ? #somme
    return ? #calcul de la moyenne
```

Définir de la même façon la fonction `ecart_type(L)` qui calcule et renvoie l'écart type σ des valeurs de la liste `L` de valeur moyenne $\overline{x}$. On rappelle que

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \overline{x})^2}$$

N'oubliez pas de documenter votre fonction.

```
def ecart_type(liste):
    "Renvoie l'écart type des valeurs de la liste" #documentation de la
    fonction
    N=? #nombre de valeurs dans la liste
```

```
    moy=? #calcul de la moyenne
    somme_carre=? # initialisation de la somme des carrés
    for valeur in liste:
        somme_carre=?
    return ? #calcul de l'écart type
```

3. Sauvegardez votre module et importez le depuis votre programme principal en ajoutant `import Biblio-statistiques as stats` dans le paragraphe `## Bibliothèques` de votre fichier principal `TP07_Modules.py`

III Mise en application

Connaissez-vous l'application **Phyphox** ? Elle transforme votre smartphone en véritable laboratoire, vous pourriez vous prendre pour Thomas Pesquet !
Bon, restons sur Terre et mesurons l'accélération de pesanteur g au niveau du sol.

En utilisant quelques secondes les accéléromètres d'un téléphone immobile et l'application **Phyphox** on a récupéré le fichier de données au format csv (comma separated values) `Raw_Data.csv` dont voici les premières lignes :

```
"Time (s)","Acceleration x (m/s^2)","Acceleration y (m/s^2)","Acceleration z (m/s^2)","Absolute acceleration (m/s^2)"
1.608458348E-3,1.062789917E-2,1.924996948E-1,9.764494629E0,9.766397721E0
1.162745839E-2,1.197509766E-2,2.007325745E-1,9.772278442E0,9.774347187E0
2.164645842E-2,6.137237549E-3,1.989363098E-1,9.755662994E0,9.757693056E0
...
```

Vous pouvez récupérer le fichier à l'adresse ci-dessous et l'enregistrer dans le même répertoire que `TP07_Modules.py`
https://www.pcsi2.net/fabert/wp-content/uploads/physique/Raw_Data.csv

III.1 Récupération des données utiles

On cherche ici à déterminer la valeur mesurée de l'accélération de pesanteur g sous la forme

$$g = \hat{g} + u(g)$$

avec $\hat{g}$ le meilleur estimateur (la valeur moyenne $\overline{g}$) et $u(g)$ l'incertitude type (écart type σ).
Les données utiles apparaissent à partir de la seconde ligne et en dernière position ("Absolute acceleration").

Question 5. en complétant le code suivant, importez les données.

```
fic = open("Raw_Data.csv", "r") # Ouverture du fichier texte en mode lecture
lignes = fic.readlines() #retourne la liste de toutes les lignes
fic.close() # Fermeture du fichier texte

n=len(lignes)
Lt,Lgx,Lgy,Lgz,Lg=[],[],[],[],[] #création des listes de données vides

for i in range(1,n): #on ne tient pas compte de la première ligne
    ligne=lignes[i]
    t,gx,gy,gz,g=ligne.split(?) #découpe de chaque ligne, on récupère les données séparées par des ','
```

```
    Lt.append(float(t)) #et ajout des données dans les listes initialement vides.
    Lgx.append(float(gx))
    Lgy.append(float(gy))
    Lgz.append(float(gz))
    Lg.append(float(?))
```

Dans la console, vérifiez que Lg est bien une liste contenant les valeurs de g mesurées par le téléphone.

III.2 Traitement des données

On va ici utiliser les fonctions de notre module stats pour calculer le meilleur estimateur et l'écart type sur la grandeur mesurée.

Question 6. calculez $\overline{g}$ et $u(g)$ puis affichez la valeur de g mesurée.

```
print("g=",round(stats.moyenne(Lg),4)," m/s^2")
print("u(g)=",?," m/s^2")
```

Réponse attendue : $g = 9.7587 \pm 0.0059$ m.s^{-2} si on conserve deux chiffres significatifs sur $u(g)$.

Remarque : en travaillant avec des tableaux (array) plutôt qu'avec des listes on aurait pu utiliser les méthodes `array.mean()` et `array.std()` pour effectuer ces calculs, c'est la méthode que nous avons utilisé en TP de physique. On pourrait par exemple taper dans la console :

```
tableau=np.array(Lg) #conversion liste en tableau
tableau.mean()
tableau.std(ddof=1)
```

III.3 Représentions graphiques

Question 7.

1. Représentez l'évolution temporelle g(t) en complétant le code ci-dessous

```
plt.figure() #création de la fenêtre graphique
plt.plot(?,?) #trace g(t)
plt.title("Accélération de pesanteur")
plt.xlabel("t en seconde")
plt.ylabel("g en m.s$^{-2}$")
#plt.ylim(0,10)
plt.show()
```

   Remarque : les parties de chaines de caractères entre $ sont du code LaTeX.

2. En décommentant l'instruction `plt.ylim(0,10)` avant `plt.show()`, remarquez que, relativement à sa valeur moyenne, $g(t)$ varie peu au cours du temps.
3. Pour visualiser cette dispersion des valeurs de g autour de $\overline{g}$, il est utile d'afficher l'histogramme des valeurs de g à l'aide de l'instruction `plt.hist(Lg)`. L'option `bins` permet de régler le nombre de barres qui apparaissent sur le diagramme.

```
plt.figure() #création de la fenêtre graphique
plt.title("Accélération de pesanteur")
plt.xlabel("Densité de probabilité")
plt.ylabel("g en m.s$^{-2}$")
plt.hist(Lg,bins='auto')
plt.show()
```

   Vous remarquerez la forme gaussienne de la distribution.