1 Principes généraux

Exercice 1 – *PGCD*

1. On veut calculer le Plus Grand Commun Diviseur de deux entiers. Par exemple, le PGCD de 18 et 42 est 6 ($18 = 3 * 6$ et $42 = 7 * 6$).
2. La bonne méthode est d'utiliser l'algorithme d'Euler (qui est basé sur le théorème de Gauss).
3. Le fichier `pgcd.py` contient du code permettant de saisir au clavier deux nombres, et qui appelle la fonction `pgcd(nombre1,nombre2)`.

1. Écrivez cette fonction et testez-la pour quelques valeurs.

Rappel : Bon, alors, comment ça marche l'algorithme d'Euler, déjà ?

- Si b divise a, alors $pgcd(a, b) = b$
- Sinon, on peut écrire $a = bt + r$, et alors $pgcd(a, b) = pgcd(b, r)$

Indice : Affectation multiple en Python.

- En Python, on peut affecter des valeurs simultanément à plusieurs variables. C'est pratique pour permuter deux valeurs : `a,b = b,a`

Exercice 2 – *Fonction factorielle*

$Factorielle(n)$ est définie de la façon suivante :

- $Factorielle(0) = 1$
- $Factorielle(n) = n * Factorielle(n - 1)$ si $n > 0$

Une autre définition possible est :

- $Factorielle(0) = 1$
- $Factorielle(n) = 1 * 2 * \cdots * n$ si $n > 0$

1. Écrire une fonction récursive calculant la factorielle d'un entier.
2. Écrire une fonction calculant la factorielle d'un entier sans faire appel à la récursion.

2 Boucles

Exercice 1 – *Boucles*

Reprenons notre liste avec les jours de la semaine. Écrire une série d'instructions affichant les jours de la semaine (utiliser une boucle `for`) ainsi qu'une autre série d'instructions affichant les jours du week-end (utiliser une boucle `while`).

Exercice 2 – *Tests*

Reprenons notre liste `semaine` ; en utilisant une boucle, écrivez chaque jour de la semaine ainsi que les messages suivants :

1. 'Au travail !' s'il s'agit du Lundi au Jeudi,
2. 'Chouette c'est Vendredi' s'il s'agit du Vendredi,
3. 'Arggh, ce week-end je dois préparer une présentation pour lundi !' s'il s'agit du week-end.

3 Listes

Exercice 1 – *Indiçage de listes*

1. Constituer une liste `'semaine'` contenant les 7 jours de la semaine. À partir de cette liste, comment récupérer les 5 premiers jours de la semaine d'une part, et ceux du week-end d'autre part (utiliser l'indiçage) ? Chercher un autre moyen pour arriver au même résultat (en utilisant un autre indiçage).
2. Créer quatre listes `hiver, printemps, ete, automne` contenant les mois correspondant à ces saisons. Créer ensuite une liste `saisons` contenant les sous-listes `hiver, printemps, ete, automne`. Prévoir ce que valent les variables suivantes, puis vérifier dans l'interpréteur :
   (a) `saisons[2]`
   (b) `saisons[1][0]`
   (c) `saisons[1:2]`
   (d) `saisons[:][1]`

Comment expliquez-vous ce dernier résultat ?

Exercice 2 – *Manipulation de listes*

1. Écrire le code PYTHON permettant d'obtenir la liste `list2` à partir de `list1`

```
list1 = ['a', 'b', 'c', 'd']
list2 = [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

2. Écrire le code PYTHON permettant d'obtenir la liste `list3` à partir de `list1` et `list2`. Utiliser `range` pour définir `list1` et `list2`.

```
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8, 9, 10]

list3 = [(1, 6), (2, 7), (3, 8), (4, 9), (5, 10)]
```

3. Écrire le code PYTHON permettant d'obtenir les listes `list2` et `list3` à partir de `list1`.

```
list1 = [(1, 6), (2, 7), (3, 8), (4, 9), (5, 10)]

list2 = [1, 2, 3, 4, 5]
list3 = [6, 7, 8, 9, 10]
```

Exercice 3 – *Compréhension de listes*

1. Construire une liste `list1` de nombres entiers allant de 11 à 22.
2. Construire une liste `list2` à partir de `list1` contenant le double de chaque élément de `list1`.
3. Construire une liste `list3` à partir de `list1` ne contenant que les éléments pairs.

Exercice 4 – *Crible d'Eratosthène*

Le crible d'Eratosthène permet de trouver tous les nombres premiers plus petits que N.

1. Construire une liste d'entiers entre 0 et N appelée crible.

2. Mettre les deux premiers éléments de la liste à None (0 et 1 ne sont pas premiers).
3. Pour chaque élément de la liste :
    - S'il est égal à `None`, ne rien faire.
    - Sinon, c'est un nombre premier, remplacez tous ses multiples par `None`.

Indices :

- Utilisez la fonction `range` pour obtenir tous les multiples d'un nombre.

1. Écrivez une fonction `prime` qui prend un paramètre `n` (qui doit être un entier) et qui retourne la liste des nombres premiers plus petits ou égaux à `n`.

4 Dictionnaires

Exercice 1 – *Manipulation de dictionnaires*

1. Fabriquer un dictionnaire qui aura pour clé les prénoms et pour valeur les âges associés à partir des 2 listes suivantes :

```
prenoms = ['arthur', 'babar', 'celeste']
ages = [5, 25, 27]
```

2. Écrire une fonction qui permet d'inverser un dictionnaire (que les clés deviennent les valeurs et les valeurs deviennent les clés). On supposera que les valeurs sont uniques.

3. Écrire le code PYTHON permettant d'obtenir le dictionnaire `d2` à partir de `d1`.

```
d1 = {
    ('paris', 'population'): 2500000,
    ('paris', 'coordoonnees'): (48.51, 2.21),
    ('paris', 'superficie'): 86.9,
    ('paris', 'arrondissements'): 20,
    ('besancon', 'population'): 120000,
    ('besancon', 'coordonnes'): (47.14, 6.01),
    ('besancon', 'superficie'): 65,
    }

d2 = {
    'paris': {'population': 2500000,
              'coordonnes': (48.51, 2.21),
              'superficie': 86.9,
              'arrondissements': 20,
              },
    'besancon': {'population': 120000,
                 'coordonnes': (47.14, 6.01),
                 'superficie': 65,
                 },
    }
```

5 Références et objets modifiables

Exercice 1 – *Gestion des références en* PYTHON

1. Que valent les variables `a` et `b` après les instructions suivantes ?

```
a = 1
b = a
a = 3
```

2. Que valent les variables `list1` et `list2` après les instructions suivantes ?

```
list1 = [1, 2, 3]
list2 = list1
list2[0] = 4
```

3. Que valent les variables `s1` et `s2` après les instructions suivantes ?

```
s1 = "abc"
s2 = s1
s1 = s1 + "def"
```

4. Que valent les variables `list1` et `list2` après les instructions suivantes ?

```
list1 = [1, 2, 3]
list2 = list1
list1 = list1 + [4, 5, 6]
```

5. Que valent les variables `s1` et `s2` après les instructions suivantes ?

```
s1 = "abc"
s2 = s1
s1 += "def"
```

6. Que valent les variables `list1` et `list2` après les instructions suivantes ?

```
list1 = [1, 2, 3]
list2 = list1
list1 += [4, 5, 6]
```

7. Que valent les variables `list1`, `list2` et `list3` après les instructions suivantes ?

```
list1 = [1, 2, 3]
list2 = ["foo", "bar"]
list3 = list2
list1.append(list2)
list3.append("baz")
```

8. Que valent les variables `list1`, `list2` et `list3` après les instructions suivantes ?

```
list1 = [1, 2, 3]
list2 = ["foo", "bar"]
list1.append(list2)
list3 = list1[3]
list3.append(789)
```

9. Que valent les variables `d1`, `d2` et `list1` après les instructions suivantes ?

```
list1 = [1, 2, "truc"]
d1 = {"toto": "bar"}
d2 = {"liste": list1}
d2['dico'] = d1
d2['liste'].append(list1[0])
d2['dico']['toto'] = "foo"
d1['titi'] = 12
```

6 Objets chaînes de caractères

Exercice 1 – *Manipulation de chaînes de caractères*

1. Écrire le code PYTHON permettant d'obtenir la chaîne `s2` à partir de la chaîne `s1`.

```
s1 = "bonjour    tout le      monde  \n"
s2 = "bonjour tout le monde"
```

2. Écrire une fonction `xml_escape(string)` qui remplace les caractères spéciaux du XML par leur entité équivalente dans la chaîne donnée en argument.

Caractère	Entité équivalente
"	`&quot;`
'	`&#39;`
<	`&lt;`
>	`&gt;`
&	`&amp;`

Exercice 2 – *Manipulation de fichiers et de chaînes de caractères*

Nous souhaitons écrire un programme qui permette de compléter un texte à trous contenu dans un fichier avec les données contenues dans un autre fichier.

1. Écrire à la main dans un fichier le petit texte suivant :

```
Bonjour <monsieur>,
Je ne suis pas disponible pour le moment car je suis une formation
sur le langage de programmation <langage>. Je serai rentré
le <date_retour>, et vous recontacterai dès que possible.

<formule_salutations>
<prenom> <nom>.
```

Dans ce texte, les trous sont des identifiants entourés des caractères < et >. Pour simplifier, les identifiants ne doivent pas contenir d'espaces, et on suppose que le fichier de valeurs est bien formé.

2. Utilisez le fichier `hole_values.txt` contenant les valeurs à utiliser pour remplir les trous. Les lignes de ce fichier seront au format `identifiant=valeur` avec un et un seul identifiant par ligne.

3. Écrire un programme qui complète le texte à trous (`text_with_holes.tx`) grâce au fichier de valeurs, et sauve le résultat dans un autre fichier. Comparez le résultat avec le contenu du fichier `filled_files.txt` fourni.

Exercice 3 – *Statistiques d'un texte*

On cherche à avoir un petit résumé des caractéristiques d'un texte suivantes :

- Nombre de mots.
- Nombre de lettres.
- Histogramme de répartition des lettres (on ignorera les lettres non-ASCII).

Vous pourrez utilisez les fichiers `keats.txt` ou `endymion.txt` fournis, ou encore un extrait d'un article Wikipedia ou d'une docstring d'un module de la bibliothèque standard de PYTHON.

L'histogramme sera représenté en graphique « ASCII » tel que :

```
a: -------
b: -
c: --
d: -
e: ----------
...
```

Le nombre de colonnes d'affichage de l'histogramme sera un paramètre passé à la fonction d'affichage de l'histogramme.

1. Écrivez la fonction de calcul du nombre de mots du texte passé en paramètre.

2. Écrivez la fonction de calcul du nombre d'occurrences de chaque lettre [a-z] dans le texte passé en paramètres. Cette fonction doit renvoyer un dictionnaire contenant les lettres comme clefs, et les nombres d'occurrences comme valeurs. On pourra rendre cette fonction sensible à la casse ou non via un paramètre booléen (optionnel, faux par défaut) de la fonction.

3. Écrivez la fonction de calcul de l'histogramme des lettres du texte passé en paramètre. Cette fonction renverra une liste de 26 valeurs (une par lettre) réelles, en pourcentage du nombre total de lettres du texte. Le compte est donc indépendant de la casse.

4. Écrivez la fonction d'affichage de l'histogramme. Pour ceux qui sont plus à l'aise, faites en sorte que l'affichage soit « vertical » et non « horizontal ». Cette fonction prendra un argument optionnel (60 par défaut) permettant d'indiquer le nombre de colonnes que doit occuper la lettre la plus fréquente.

5. Écrivez la fonction d'affichage de l'ensemble des statistiques du texte lu dans un fichier dont le nom est passé en paramètre, qui produira un affichage tel que :

```
Statistiques du texte "nom_du_fichier.txt"
==========================================

Nombre de mots :    XX
Nombre de lettres : XX

Répartition des lettres :

a: -------
b: --
...
z: -
```

7 La base du modèle objet

Exercice 1 – *Reprise d'un système de facturation existant*

Ouvrez `invoice.py`. La méthode `Invoice.Append` accepte un objet en argument qui correspond à une ligne de facturation :

1. Écrivez une classe de base qui correspond à l'interface utilisée par `InvoiceItem`.

2. Ecrivez les classes des lignes de facturation pour des clous et des planches de bois.

Exercice 2 – *Écriture de classes permettant de manipuler des géometries 2D*

1. Écrivez une classe `Point` permettant de représenter un point de coordonnées (x, y). Écrivez une méthode renvoyant la distance avec un autre point.

2. Écrivez des classes pour représenter différentes formes (cercle, carré, triangle, polygone, ...) dérivant d'une classe abstraite `Figure`.

3. La classe `Figure` sera composée d'une suite de point.

4. Chaque classe devra avoir une méthode renvoyant sa position et une autre son périmètre.

8 Gestion des exceptions

Exercice 1 – *Traitement des exceptions standard*

1. Écrivez une fonction `input_integer(question)` qui prend une chaîne en argument.

- La fonction doit afficher la chaîne et attendre que l'utilisateur saisisse une réponse, puis transformer la réponse en un entier, pour finalement retourner cet entier à la fonction appelante.
  - Si la conversion de la chaîne en entier échoue, il faut donner un message informatif à l'utilisateur et reposer la question jusqu'à ce que cela fonctionne.

2. Appelez la fonction pour demander l'âge de l'utilisateur.

- Traiter l'appui sur `Ctrl-C` et `Ctrl-D` par l'utilisateur (le programme doit se terminer sans afficher de message d'erreur).

3. Modifiez `input_integer(question)` pour utiliser une exception personnalisée si la valeur est en dehors des limites passées en arguments.

9 Introspection

Exercice 1 – *Introspection*

1. Écrire une fonction similaire à la fonction `help` en vous aidant du module `inspect`

- Cette fonction prend en argument un module, une classe, une fonction... et affiche le maximum d'information sur cet objet
- N'hésitez pas à vous servir de l'interpréteur interactif pour découvrir ce dont vous avez besoin

10 Entrée et sortie standard

Exercice 1 – *Manipulation de fichiers*

1. Écrire le code Python permettant d'ouvrir un fichier et de générer le fichier renversé, c'est-à-dire avec la dernière ligne en premier et la première en dernier.

Exercice 2 – *Script d'envoi d'emails*

1. Écrire un script Python permettant d'envoyer un fichier à plusieurs personnes en utilisant la commande :

```
mail -s subject email < fichier
```

1. Pensez à utiliser la fonction `os.system`, le module `optparse` ou encore le programme de l'exercice sur les chaînes permettant de valider une adresse électronique.
2. Pour simuler l'envoi on utilisera la commande système `echo`.

Exercice 3 – *Lecture de fichier .ini*

1. Écrire une fonction de lecture d'un fichier au format `.ini` . On veut par exemple que la lecture du fichier suivant (utilisez le fichier `testconf.ini`) :

```
# debut
x = 1
y = 2

[sect1]
# commentaire
x=2
y=abcd

[sect2]
b=hello
```

produise un dictionnaire :

```
{ 'global' : { 'x' : '1', 'y' : '2' },
  'sect1' : { 'x' : '2', 'y' : 'abcd' },
  'sect2' : { 'b' : 'hello'},
}
```

Exercice 4 – *Lecture et écriture de fichier texte*

1. Faire un script simple qui va lire un fichier texte (dont le nom aura été passé en paramètre de la ligne de commande) contenant des mesures de tensions électriques (une par colonne, les colonnes

sont séparées par des tabulations), dont la première ligne contient le nom du capteur pour chaque colonne, et qui produira un dictionnaire, dont les clefs seront les noms des capteurs et dont les valeurs seront les listes des valeurs numériques lues dans le fichier (donc des listes de réels). Exemple :

```
Vcc     Vpp     Vbe1     V1
12.1    5.02    0.67     3.1
12.1    5.03    0.67     3.15
...
```

Pensez à utiliser le module `csv` et le fichier `sensor.txt` pour tester votre script.

2. Générer à partir de ce dictionnaire un fichier Python qui pourra par la suite être importé comme un simple module contenant un dictionnaire nommé "mesures".

3. À partir de ce dictionnaire, calculer les valeurs moyenne, min et max pour chaque capteur, et générer un fichier (dont le nom sera celui du fichier lu juste avant affecté du suffixe ".stats") ressemblant à :

```
STATISTIQUES DU FICHIER 'monfichierdemesure.txt'
------------------------------------------------

+--------+--------+-------+---------+
|Capteur |   min  |  max  | moyenne |
+========+========+=======+=========+
|Vcc     |  11.97 | 12.65 |   12.05 |
+--------+--------+-------+---------+
...
+--------+--------+-------+---------+
```

4. Toujours à partir de ce dictionnaire, générer pour chaque série de mesures les quantiles de leur distribution de valeurs à 5%, 10%, etc.

11 Expressions régulières

Exercice 1 – *Vérification d'adresses email*

1. Ecrire une fonction python dans le fichier `checkemail.py` dont l'objectif est de vérifier la validité d'une adresse électronique.

- On dira qu'une adresse est valide si :
  1. La partie à gauche de l'arobase n'est composée que de caractères, chiffres, tirets et points.
  2. La partie à droite de l'arobase est une suite d'identifiants séparés par des points, ces identifiants ne pouvant comporter que des caractères alphanumériques et des tirets.
- Pour les spécifications exactes voir la RFC 822.

12 Autres modules fréquemment utilisés

Exercice 1 – *Commande find*

On veut créer une commande similaire à la commande Unix `find`

1. Écrire une fonction `search(directory, regexp)` qui va parcourir récursivement le répertoire passé en 1er argument et renvoyer les fichiers dont le nom correspond à l'expression rationnelle passée en 2eme argument
2. Écrire une fonction `main` qui va proposer une interface en ligne de commande à cette fonction
3. Ajouter une option '-i/–case-insensitive' pour rendre la recherche insensible à la casse
4. Ajouter une option '-e/–exec <CMD>' pour exécuter la commande donnée sur chaque fichier trouvé

Exercice 2 – *Télécharger des données d'un serveur distant avec urllib*

On veut automatiser l'extraction de données d'un serveur web effectuant des calculs à distance. Par exemple, on propose d'utiliser ici le serveur X0h (calcul de facteurs de dispersion des rayons X par différents matériaux) : `http://x-server.gmca.aps.anl.gov/cgi/WWW_form.exe?template=x0h_form.htm`

L'adresse d'une requête spécifique sera par exemple (en remplissant le formulaire ci-dessus) : `http://x-server.gmca.aps.anl.gov/cgi/X0h_form.exe?xway=2&wave=12.4&coway=0&code=Silicon&i1=1&i2=1&i3=1&df1df2=-1&modeout=1`

1. Écrire une fonction create_url(baseurl, paramdict) qui crée l'url de la requête à partir d'un dictionnaire de paramètres (xway, wave, coway...)
2. Écrire une fonction fetch_data(fullurl) qui renvoie le contenu de la page web avec urllib
3. Écrire une fonction parse_result qui extrait un des résultats de calcul (par exemple mu0)
4. Avancé : utiliser les expressions régulières pour extraire plusieurs données dans un dictionnaire en sortie

13 Outils d'analyse de code

Objectif : Savoir utiliser les outils d'analyse statique du code

Exercice 1 – *Utilisation de Pylint*

1. Lancer pylint quelques fichiers sources que vous avez écrits et essayer de corriger quelques erreurs ou avertissements.

2. Certains messages sont sans doute superflus. Selon les cas, les désactiver globalement ou localement (avec une portée précise dans le module).

14 Tests unitaires

Exercice 1 – *Tests unitaires du programme de validation d'adresses email*

1. Écrivez les tests pour le programme de validation d'adresses email

Exercice 2 – *Tests unitaires d'un programme de calcul de poussée*

- Le module `rocket.py` calcule la poussée des réacteurs d'une fusée en fonction de ses coordonnées.
- Le fichier `rocket_thrust.txt` contient un tableau :
  - `temps, coord x, coord y, poussée x, poussée y`

  décrivant la poussée attendue pour une trajectoire donnée
- Le fichier `rocket_trajectory.txt` contient un tableau de triplets `(temps, posx, posy)`.

  1. Écrivez les tests permettant de vérifier que :

  1. l'algorithme de calcul de poussée est correct ;
  2. la trajectoire décrite dans le fichier `rocket_trajectory.txt` peut être suivie.

15 Appeler des bibliothèques C

Exercice 1 – *Interfaçage avec du code C*

1. Écrire une fonction factorielle en C, en respectant l'API C de Python. Vérifier qu'on peut bien l'importer depuis python, et que le résultat est correct.

2. Que se passe-t-il si vous utilisez votre fonction C depuis python pour calculer la factorielle de 50 ? Pourquoi ?

16 Glade

Exercice 1 – *GTK - Glade*

1. Réaliser une fenêtre d'écriture d'email

- La fenêtre aura 3 champs textes (*Destinaires*, *Cc*, et *Sujet*)
- Il faudra évidemment prévoir un champ pour pouvoir saisir le corps du message
- Il y aura également un bouton *Envoyer* qui sera grisé si une ou plusieurs adresses de destinataires sont invalides. Lorsqu'on cliquera sur ce bouton, on se contentera de faire un `print` à l'écran indiquant les caractéristiques principales du message[1]
- Prévoir un bouton *Annuler*

17 Manipulation de tableaux numériques

Exercice 1 – *Créer les tableaux suivants le plus simplement possible*

- $A = \begin{pmatrix} 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 2. \\ 1. & 6. & 1. & 1. \end{pmatrix}$

[1] Si vous souhaitez réellement envoyer un message, il vous faudra probablement utiliser les modules `smtplib` et `email` de la librairie standard

- $B = \begin{pmatrix} 0. & 0. & 0. & 0. & 0. \\ 2. & 0. & 0. & 0. & 0. \\ 0. & 3. & 0. & 0. & 0. \\ 0. & 0. & 4. & 0. & 0. \\ 0. & 0. & 0. & 5. & 0. \\ 0. & 0. & 0. & 0. & 6. \end{pmatrix}$

Exercice 2 – *Statistiques*

1. Créer un tableau en ouvrant le fichier `women_percentage.txt`. Quelle est la taille de ce tableau ?
2. Les colonnes correspondent aux années 2006 à 2001. Créer un tableau d'entiers correspondant à ces années.
3. Les différentes lignes correspondent aux organismes de recherche dont les noms sont stockés dans le fichier `organisms.txt`. Créér un tableau de chaînes de caractères contenant ces noms à partir du fichier.
4. Vérifier que le nombre de lignes des données est égal au nombre d'organismes.
5. Quel est le pourcentage maximal de femmes parmi les organismes pour chaque année.
6. Créer un tableau contenant la moyenne temporelle pour chaque organisme.
7. Quel organisme a le plus grand pourcentage de femme en 2004 ?
8. Créer un tableau qui contient les organismes où le plus grand taux de femmes est trouvé pour les différentes années.

Exercice 3 – *Filtrage d'une image*

On dispose d'un tableau bidimensionnel `I` de taille 10×10 et on souhaite calculer le tableau `I2` en utilisant la transformation suivante :

```
I2[i,j] = 0.25 * (I[i-1, j] + I[i+1, j] + I[i, j-1] + I[i, j+1]
```

1. Créer le tableau `I` en utilisant le module `random` de `NumPy`.
2. Écrire une fonction réalisant la transformation.

Exercice 4 – *Calculer π par Monte-Carlo*

Une façon notoirement inefficace de calculer π est de lancer des fléchettes dans une cible ronde fixée sur un support carré (le rond est inscrit dans le carré) les yeux bandés.

En faisant l'hypothèse qu'avec les yeux bandés, chaque fléchette tirée qui atteint le support de la cible aura effectivement été tirée au hasard (selon une loi uniforme), on peut aisément estimer la valeur de π en comptant le nombre de fléchettes qui sont dans la cible et le nombre de fléchettes qui ont atteint le support mais ne sont pas dans la cible.

1. Écrire la formule qui donne l'estimation de la valeur de π en fonction des paramètres du problème.

2. Écrire une fonction qui prend un argument N, le nombre de tirs de fléchettes que l'on veut simuler, et qui renvoie l'estimation de la valeur de π.

Il est autorisé de ne simuler que les tirs de fléchettes « réussis ». On entend par « réussi » tout tir qui atteint le support ou la cible.

3. Améliorer la fonction pour qu'elle accepte un second argument (optionnel), le pas p des dates d'observation de l'estimateur. La fonction doit alors renvoyer un tableau `NumPy` de dimension N/p contenant les différentes estimations de π calculées au fur et à mesure de l'évolution de la simulation (donc au fur à mesure du nombre de tirs de fléchette simulés, par pas de p tirs).

4. Tracer la courbe d'erreur en fonction du temps de simulation.

Pour estimer l'erreur, il est autorisé de comparer le résultat calculé avec la valeur disponible dans le module `NumPy`.

Conclure sur la méthode (et la pertinence de s'autoriser à estimer l'erreur en utilisant l'approximation de π disponible dans le module `NumPy`).

Exercice 5 – *Mesures expérimentales*

Le fichier `volume_experiment_raw_data.dat` contient des mesures de pression et de température au sein d'un volume discrétisé en petits cubes à différents instants d'une expérience.

Les lignes commençant par `Time step` donnent la valeur des instants t et annoncent une série de mesures sur tout le volume pour cette valeur de t. Les lignes de mesure commencent par les coordonnées du petit cube (trois entiers x, y et z) suivies des valeurs de la pression P et de la température T (deux flottants).

On veut calculer la moyenne de la pression et de la température pour un petit cube donné au cours de l'expérience, ainsi que la pression et la température moyenne de l'ensemble du volume à chaque pas de temps.

1. Lire les données dans un tableau numpy à 5 axes :
`[temps, x, y, z, P, T]`
Par exemple, si l'expérience contient 4 pas de temps et que le volume a été discrétisé en $5x5x5$ petits cubes, la forme (`shape`) du tableau doit être : `(4, 5, 5, 5, 2)`.

2. Calculer les moyennes demandées en extrayant des tranches du tableau de données créé précédemment et écrire les résultats dans des fichiers au format semblable à celui reçu en entrée.

3. Pour les courageux, faire la même chose sans `NumPy`, en n'utilisant que les structures de données de PYTHON (listes, dictionnaires, tuples, etc.)

18 Stockage de données

Exercice 1 – *Utilisation de NetCDF*

Vous devez échanger des données avec une application maison en Fortran, tournant sur un serveur distant Le format compris et produit par l'application est le suivant :

position	longueur	description
0	1	précision : 0 pour flottant 32 bits, 1 pour flottant 64 bits
1	2	entier non signé donnant le nombre de matrices dans le fichier
3	4	entier : nombre de colonnes de la première matrice
7	4	entier : nombre de lignes de la première matrice
11	<3>*<4>*taille des données	données de la première matrice par colonne d'abord (convetion FORTRANT)
...	...	<3> à <5> répétées <2> fois

1. Écrire une fonction `save_custom(nomfichier, liste_tabs)` qui sauve les tableaux de la liste en utilisant le format ci-dessus

2. Écrire une fonction `load_custom(nomfichier)` qui renvoie une liste de tableaux

3. Si le serveur distant est une machine Sparc et que vous travaillez sur un PC, que devez vous faire ?

4. Ajoutez un argument `mode` à `save_custom`. Si le mode est `'a'` et que le fichier existe, la fonction devra efficacement ajouter les matrices à la fin du fichier et mettre à jour le nombre de matrices