Modélisation numérique en physique

Séquence 2 : Python intermédiaire

Légende :
☆☆ : Élément fondamental, à noter ++
☆ : A noter

Cours et calepins : https://phys-mod.github.io/source/pages/part-python-intermediaire.html

Diata Traore (diata.traore@sorbonne-universite.fr)

Objectifs de la séquence :

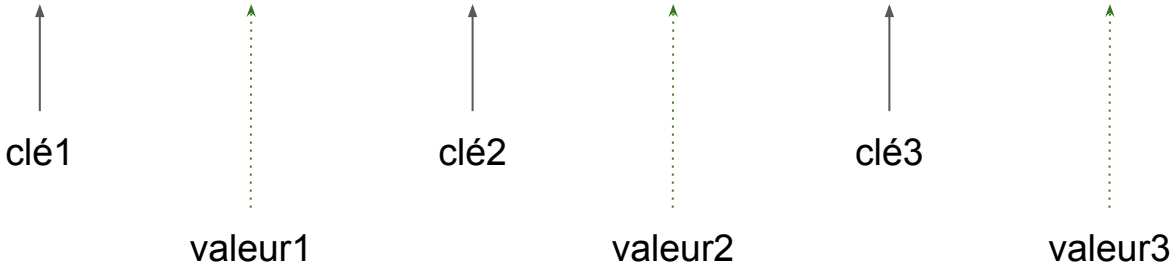
- *Créer et manipuler*
  - *les dictionnaires.*
  - *l'objet DataFrame du module Pandas.*
  - *les objets Series et Index du module pandas.*
- *Décrire et utiliser les opérateurs de comparaisons sur les booléens.*
- *Extraire et filtrer des tableaux Numpy et des DataFrame en utilisant des conditions.*
- *Ecrire la syntaxe et créer des structures conditionnelles et itératives (if, for et while)*
- *Ecrire la syntaxe d'une fonction Python, décrire le fonctionnement des arguments d'entrée et de sortie, et créer des fonctions simples.*
- *Utiliser l'entrée et la sortie standard, écrire et lire des fichiers avec un format défini.*

(D'après la liste des objectifs définis par l'équipe pédagogique lors de la création du format actuel de l'ue.)

Objectif #1 : créer et manipuler les **dictionnaires** et les **DataFrame** du module Pandas

☆☆ Syntaxe d'un dictionnaire :

```
mon_dictionnaire = { allemagne : berlin , autriche : vienne , belgique : bruxelles , }
```



☆ Afficher la capitale de l'Autriche :

```
mon_dictionnaire['hongrie']
```

☆ Ajouter la paire 'bulgarie', 'sofia' au dictionnaire :

```
mon_dictionnaire['bulgarie'] = 'sofia'
```

Vérifier qu'une clé se trouve dans le dictionnaire :

```
'bulgarie' in mon_dictionnaire
```

Supprimer 'bulgarie' de mon_dictionnaire :

```
del(mon_dictionnaire['bulgarie'])
```

Objectif #1 : créer et manipuler les **dictionnaires** et les **DataFrame** du module Pandas

☆☆ Importer le module Pandas :

```
import pandas
```

```
dic_europe = {
    "pays" : np_pays,
    "capitale" : np_capitales,
    "population" : np_population,
    "date d'adhésion" : np_date
}
```

☆☆ Créer et afficher un DataFrame à partir du dictionnaire dic_europe

```
df_europe = pd.DataFrame(data = dic_europe)
```

	pays	capitale	population	date d'adhésion
0	allemagne	berlin	82162000	1957-01-01
1	autriche	vienne	8700471	1995-01-01
2	belgique	bruxelles	11289853	1957-01-01
3	bulgarie	sofia	7153784	2007-01-01
4	chypre	nicosie	848319	2004-01-01
5	croatie	zagreb	4190669	2013-01-01
6	danemark	copenhague	5659715	1973-01-01
7	espagne	madrid	46438422	1986-01-01
8	estonie	tallinn	1315944	2004-01-01
9	finlande	helsinki	5401267	1995-01-01
10	france	paris	66661621	1957-01-01
11	grèce	athènes	10793526	1981-01-01

Afficher les 3 premiers éléments de mon DataFrame :

```
df_europe.head(n=3)
```

Afficher les 3 derniers éléments de mon DataFrame :

```
df_europe.head(n=2)
```

Objectif #1 : créer et manipuler les **dictionnaires** et les **DataFrame** du module Pandas

☆ Afficher les informations de la ligne 9 :

```
print(df_europe.loc[9])
```

```
pays                              finlande
capitale                          helsinki
population                         5401267
date d'adhésion    1995-01-01 00:00:00
Name: 9, dtype: object
```

☆ Afficher les capitales et populations des 3 premiers pays :

```
print( df_europe.loc[ 0:2, ['capitale','population'] ] )

print( df_europe.iloc[ 0:3, [1,2] ] )
```

	capitale	population
0	berlin	82162000
1	vienne	8700471
2	bruxelles	11289853

☆ Afficher le nom du 4ème pays :

```
print( df_europe.at[3,'pays'] )
print( df_europe.iat[2,0] )
```

	pays	capitale	population	date d'adhésion
0	allemagne	berlin	82162000	1957-01-01
1	autriche	vienne	8700471	1995-01-01
2	belgique	bruxelles	11289853	1957-01-01
3	bulgarie	sofia	7153784	2007-01-01
4	chypre	nicosie	848319	2004-01-01
5	croatie	zagreb	4190669	2013-01-01
6	danemark	copenhague	5659715	1973-01-01
7	espagne	madrid	46438422	1986-01-01
8	estonie	tallinn	1315944	2004-01-01
9	finlande	helsinki	5401267	1995-01-01
10	france	paris	66661621	1957-01-01
11	grèce	athènes	10793526	1981-01-01

Objectif #1 : créer et manipuler **les dictionnaires** et les **DataFrame** du module Pandas

```
ligne_turquie = pd.Series(data=['turquie', 'ankara', 83154997, np.datetime64('2026', 'Y')],
                                        index=df_europe.columns,name=28)
```

☆☆ Ajouter ligne_turquie au DataFrame :

```
df_europe.append(ligne_turquie)
```

☆ Supprimer la ligne correspondant à la Turquie (à partir de son indice de ligne) :

```
df_europe.drop[28]
```

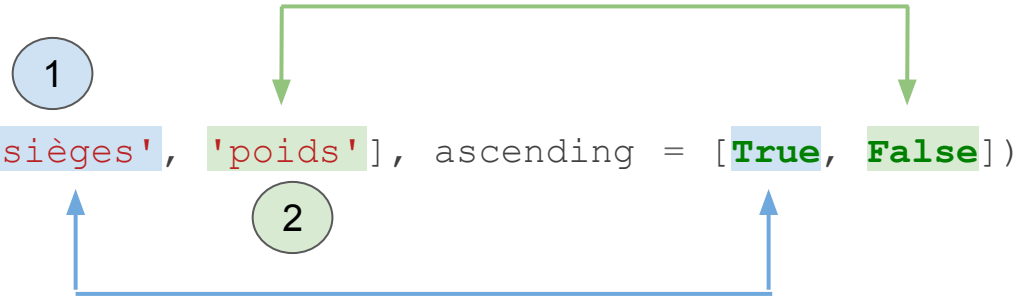
```
np_sieges = np.array([96, 18, 21, 17, 6, 11, 13, 54, 6, 13, 74, 21, 21, 11, 73, 8, 11, 6, 6, 26, 51, 21, 21, 32, 13, 8, 20])
```

☆☆ Ajouter la colonne np_sieges au DataFrame :

```
df_europe['sieges'] = np_sieges
```

☆ Trier un DataFrame :

```
df_europe.sort_values(by = ['sièges', 'poids'], ascending = [True, False])
```



Remarques :
On trie d'abord par rapport à 1 ,
Puis, par rapport à 2 .

ascending = True : ordre croissant / alphabétique

	pays	capitale	population	date d'adhésion	sièges	poids
19	malte	la valette	434403	2004-01-01	6	16.198887
18	luxembourg	luxembourg	576249	1957-01-01	6	12.211466
5	chypre	nicosie	848319	2004-01-01	6	8.295046
9	estonie	tallinn	1315944	2004-01-01	6	5.347374
16	lettonie	riga	1968957	2004-01-01	8	4.765193
26	slovénie	ljubljana	2064188	2004-01-01	8	4.545352
17	lituanie	vilnius	2888558	2004-01-01	11	4.466202

Objectif #1 : créer et manipuler les **dictionnaires** et les **DataFrame** du module Pandas

☆ Sauver un DataFrame au format .csv :

```
df_europe.to_csv("europe.csv")
```

☆ Créer un DataFrame à partir d'un .csv :

```
df_europe_from_csv = pd.read_csv("europe.csv", index_col = 0, encoding =
'utf-8')
```

☆☆ Sauver un DataFrame au format pickle :

```
df_europe.to_pickle('europe.pkl')
```

☆☆ Créer un DataFrame à partir d'un fichier pickle :

```
df_europe_from_pickle = pd.read_pickle('europe.pkl')
```

Objectif #2 : décrire et utiliser les **opérateurs de comparaisons** sur les **booléens**

☆☆ Opérateurs de comparaison :

Opérateur	Description	"Résultat"
x == y	x est égal à y	Booléens : **True** ou **False**
x != y	x est différent de y	
x > y	x est supérieur à y	
x < y	x est inférieur à y	
x >= y	x est supérieur ou égal à y	
x <= y	x est inférieur ou égal à y	

- Si x et y sont des entiers ou des flottants : comparaison des valeurs

- Si x et y sont des str : ordre alphabétique

- Si x et y sont des tableaux Numpy : comparaison terme à terme, résultat = tableau de booléens

Objectif #2 : décrire et utiliser les **opérateurs de comparaisons** sur les **booléens**

☆☆ Combinaison d'opérateurs de comparaison, les opérateurs logiques :

Opérateurs logiques	Descriptions
or	ou
and	et
not	non

☆☆ Pour les tableaux Numpy

Opérateurs logiques	Descriptions
np.logical_or(bool_1, bool_2)	ou
np.logical_and(bool_1, bool_2)	et
np.logical_not(bool_1, bool_2)	non

Objectif #3 : extraire et **filtrer des tableaux Numpy** et des DataFrame en utilisant des conditions

Inscrire les résultats affichés lors de l'exécution du script ci-dessous :

```
# creer des tableaux Numpy
A = np.array([1.4,  7.1, 9.0, 2.5, -4.6])
B = np.array([2.6, -5.7, 4.0, 2.5,  5.8])

# cond: tableau Numpy de booleens
cond = A > 4
print(cond)

# filtrer le tableau A avec le tableau cond
print(A[cond])
C=A[cond]
print(C)

# il est possible de filtrer le tableau B avec la condition sur A
print(B[cond])
```

[false,true,true,false,false]

[7.1, 9.0]

[-5.7, 4.0]

ou :

```
print( A[A > 4] )
print( B[A > 4] )
```

☆ Combiner les conditions :

```
cond = np.logical_and(A > 0, A < 5)
print( A[cond] )
```

Objectif #3 : extraire et **filtrer** des tableaux Numpy et **des DataFrame** en utilisant des conditions

```python
# import du module pandas
import pandas as pd

# lecture du fichier pickle contenant les données
df_europe = pd.read_pickle('europe.pkl')

# on extrait l'objet Series correspondant à la colonne 'population'
sr_population = df_europe['population']

# on applique la condition sur le nombre d'habitants
cond = sr_population > 50e6

# on filtre l'objet df_europe avec l'objet cond
df_europe_big = df_europe[cond]

# on affiche le tableau filtré
df_europe_big
```

	pays	capitale	population	date d'adhésion	sièges	poids
1	allemagne	berlin	82162000	1957-01-01	96	1.370336
11	france	paris	66661621	1957-01-01	74	1.301915
15	italie	rome	60665551	1957-01-01	73	1.411261

☆ Combiner les conditions :

```
df_europe[ ~(df_europe["date d'adhésion"] == np.datetime64('1957','Y') ) &(df_europe["sièges"] > 50 )
]
```

non

et

ou : |

Fin des objectifs de la séquence 2

Séquence 3 :

- *Créer et manipuler*
  - *les dictionnaires.*
  - *l'objet DataFrame du module Pandas.*
  - *les objets Series et Index du module pandas.*
- *Décrire et utiliser les opérateurs de comparaisons sur les booléens.*
- *Extraire et filtrer des tableaux Numpy et des DataFrame en utilisant des conditions.*
- *Ecrire la syntaxe et créer des structures conditionnelles et itératives (if, for et while)*
- *Ecrire la syntaxe d'une fonction Python, décrire le fonctionnement des arguments d'entrée et de sortie, et créer des fonctions simples.*
- *Utiliser l'entrée et la sortie standard, écrire et lire des fichiers avec un format défini.*

\+ Représentation graphique d'une série mathématique

Objectif #4 : écrire la syntaxe et créer des **structures conditionnelles et itératives** (if, for et while)

☆☆ Si x est strictement positif, afficher “x est strictement positif”, sinon afficher “x est inférieur ou égal à 0” :

```
x = 2
```

Objectif #4 : écrire la syntaxe et créer **des structures conditionnelles et itératives** (if, for et while)

☆☆ Si x est strictement positif, afficher “x est strictement positif”, sinon afficher “x est inférieur ou égal à 0” :

```
x = 2

if x > 0 :
   print(“x est strictement positif.”)
else :
   print(“x est inférieur ou égal à 0.”)
```

```
bouton_1 = False
bouton_2 = True

if bouton_1 and bouton_2:
    print("Les 2 boutons sont allumés")
elif bouton_1:
    print("Le bouton 1 est allumé")
elif bouton_2:
    print("Le bouton 2 est allumé")
elif bouton_1 or bouton_2:
    print("un des 2 boutons est allumé")
else:
    print("aucun des 2 boutons n'est allumé")
```

Objectif #4 : écrire la syntaxe et créer des **structures conditionnelles et itératives** (if, for et while)

☆☆ Tant que x est supérieur à 1.0, diviser x par 4.0.
☆ Ajouter une instruction qui vérifie que x est pair.

```
x = 40.0

while x > 1.0 and x%2 == 0 :
  x = x / 4.0
```

Objectif #4 : écrire la syntaxe et créer des **structures conditionnelles et itératives** (if, for et while)

☆☆ Itérer sur une variable :

Pour n compris entre 0 et 40, afficher les multiples de 4.

```
for i in range(11) :
    print(4*i)
```

☆☆ Itérer sur un dictionnaire :

```
for key, values in dic_taille_plantes.items() :
   print(key + "--" + str(values) + "cm")
```

```
dic_taille_plantes = {"gentiane"  : 7.5,
                      "campanule" : 5.8,
                      "pensée"    : 11.4}
```

```
gentiane -- 7.5 cm
campanule -- 5.8 cm
pensée -- 11.4 cm
```

☆☆ Itérer sur un tableau Numpy :

```
for i in A :
 print(i)
```

```
# importation du module numpy
import numpy as np

# creation du tableau Numpy
A = np.array([4, 6, 1])
```

Objectif #4 : écrire la syntaxe et créer des **structures conditionnelles et itératives** (if, for et while)

```python
import pandas as pd

# Création d'un dictionnaire
dic_taille_plantes = {"gentiane"  : [7.5, 7.8, 8.3, 8.4],
                      "campanule" : [5.8, 6.6, 7.4, 8.3],
                      "pensée"    : [11.4, 11.6, 11.7, 11.7]}

# Création du DataFrame
df_taille_plantes = pd.DataFrame(data = dic_taille_plantes, index = range(1, 5))
```

	gentiane	campanule	pensée
1	7.5	5.8	11.4
2	7.8	6.6	11.6
3	8.3	7.4	11.7
4	8.4	8.3	11.7

☆☆ Itérer sur les lignes d'un DataFrame :

```
for label, row in df_taille_plantes.iterrows() :
   print ("jour", label)
   print(row)
   print()
```

```
jour 1
gentiane      7.5
campanule     5.8
pensée       11.4
Name: 1, dtype: float64

jour 2
gentiane      7.8
campanule     6.6
pensée       11.6
Name: 2, dtype: float64

jour 3
gentiane      8.3
campanule     7.4
pensée       11.7
Name: 3, dtype: float64

jour 4
gentiane      8.4
campanule     8.3
pensée       11.7
Name: 4, dtype: float64
```

Objectif #4 : écrire la syntaxe et créer des **structures conditionnelles et itératives** (if, for et while)

```python
import pandas as pd

# Création d'un dictionnaire
dic_taille_plantes = {"gentiane"  : [7.5, 7.8, 8.3, 8.4],
                      "campanule" : [5.8, 6.6, 7.4, 8.3],
                      "pensée"    : [11.4, 11.6, 11.7, 11.7]}

# Création du DataFrame
df_taille_plantes = pd.DataFrame(data = dic_taille_plantes, index = range(1, 5))
```

	gentiane	campanule	pensée
1	7.5	5.8	11.4
2	7.8	6.6	11.6
3	8.3	7.4	11.7
4	8.4	8.3	11.7

☆☆ Itérer sur les colonnes d'un DataFrame

```
.
for label, col in df_taille_plantes.iteritems() :
   print ("Plante", label)
   print(col)
   print()
```

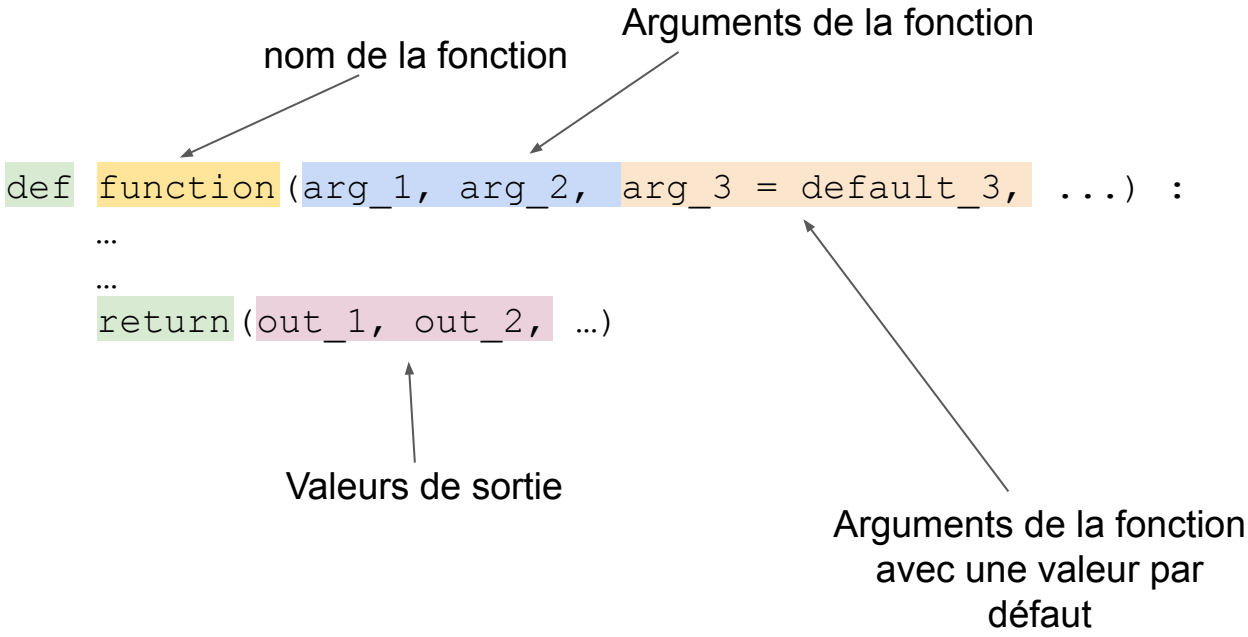
```
Plante : gentiane
1    7.5
2    7.8
3    8.3
4    8.4
Name: gentiane, dtype: float64

Plante : campanule
1    5.8
2    6.6
3    7.4
4    8.3
Name: campanule, dtype: float64

Plante : pensée
1    11.4
2    11.6
3    11.7
4    11.7
Name: pensée, dtype: float64
```

Objectif #5 : écrire la syntaxe d'une **fonction Python**, décrire le fonctionnement des arguments d'entrée et de sortie, et créer des fonctions simples

☆☆ Syntaxe générale d'une fonction python :



Objectif #5 : écrire la syntaxe d'une **fonction Python**, décrire le fonctionnement des arguments d'entrée et de sortie, et créer des fonctions simples

☆☆ Syntaxe générale d'une fonction python :

```
def function(arg_1, arg_2, arg_3 = default_3, ... ) :
    …
    …
    return(out_1, out_2, …)
```

☆☆ Appel d'une fonction python :

```
var_1, var_2, … = function(arg_1, arg2, ...)
```

Autant de variables que de valeurs en sortie

Appel de la fonction avec les valeurs des variables (attention au type)

Objectif #6 : utiliser l'entrée et la sortie standard, écrire et lire des fichiers avec un format défini

☆ Vérifier le contenu de son dossier de travail sous windows : ! dir

☆☆ Ouvrir le fichier essai.txt en mode lecture :

```
file = open( 'nom' , mode = 'r', encoding= 'uft-8')
```

☆☆ Ouvrir un fichier en mode écriture :

```
file = open( 'nom', mode = 'w', encoding= 'uft-8')
```

☆☆ Fermer un fichier :

```

```

☆☆ Ouvrir un fichier en mode écriture avec le *Gestionnaire de fichiers :*

```
with open('nom' ,mode ='w' , encoding='uft-8') as file  :
   …
```

Objectif #6 : utiliser l'entrée et la sortie standard, écrire et lire des fichiers avec un format défini

☆ Ajouter du texte à un fichier déjà existant :

```
with open('essai.txt', mode='a', encoding='uft-8') as file :
    file.write('Mon texte \n Retour à la ligne.")
```

☆ Ecraser un fichier avec un texte :

```
with open('essai.txt', mode='w', encoding='uft-8') as file :
    file.write('Mon texte \n Retour à la ligne.")
```

☆☆ Fonction permettant de lire tout ce qui est écrit au clavier (str) :

```
chaine = input("Entrer votre message avec input: \n")
```

Objectif #6 : utiliser l'entrée et la sortie standard, écrire et lire des fichiers avec un format défini

☆☆ Importer des données depuis un fichier .dat :

```
import numpy as np

data = np.loadtxt ('exoplanets.dat')
print(data[1:20][:])
```

☆☆ Importer des données en attribuant chaque colonne à un tableau 1D :

```
a, p, mp, me = np.loadtxt ('exoplanets.dat',unpack = True)
```

```
# demi-gd axe [UA]      période [j]     masse [m_jupiter]       masse étoile [m_solaire]
1.60044 672.1   11.1883 1.2
0.0761616       7.008151        0.00752338      1.2
0.519687        124.9144        0.0254263       1.2
0.736177        210.60697       0.217563        1.2
0.0530508       4.072   1.32811 1.2
1.25186 471.6   7.12332 1.17
0.996347        331.60059       0.937417        1.2
2.33328 1187.3  11.6809 1.191
0.0465021       3.35524 0.685857        1.191
0.113014        12.7204 0.0899033       1.19
0.034488        2.143634        0.984236        1.19
1.11234 392.6   1.2654  1.19
0.0702822       6.2385  0.0223552       1.19
0.0454854       3.24674 1.02109 1.19
2.60488 1405    4.7347  1.19
0.0499354       3.735433        0.508547        1.19
0.047858        3.509267        0.46004 1.187
```

exoplanets.dat

```
[[7.6161600e-02 7.0081510e+00 7.5233800e-03 1.2000000e+00]
 [5.1968700e-01 1.2491440e+02 2.5426300e-02 1.2000000e+00]
 [7.3617700e-01 2.1060697e+02 2.1756300e-01 1.2000000e+00]
 [5.3050800e-02 4.0720000e+00 1.3281100e+00 1.2000000e+00]
 [1.2518600e+00 4.7160000e+02 7.1233200e+00 1.1700000e+00]
 [9.9634700e-01 3.3160059e+02 9.3741700e-01 1.2000000e+00]
 [2.3332800e+00 1.1873000e+03 1.1680900e+01 1.1910000e+00]
 [4.6502100e-02 3.3552400e+00 6.8585700e-01 1.1910000e+00]
 [1.1301400e-01 1.2720400e+01 8.9903300e-02 1.1900000e+00]
 [3.4488000e-02 2.1436340e+00 9.8423600e-01 1.1900000e+00]
 [1.1123400e+00 3.9260000e+02 1.2654000e+00 1.1900000e+00]
 [7.0282200e-02 6.2385000e+00 2.2355200e-02 1.1900000e+00]
 [4.5485400e-02 3.2467400e+00 1.0210900e+00 1.1900000e+00]
 [2.6048800e+00 1.4050000e+03 4.7347000e+00 1.1900000e+00]
 [4.9935400e-02 3.7354330e+00 5.0854700e-01 1.1900000e+00]
 [4.7858000e-02 3.5092670e+00 4.6004000e-01 1.1870000e+00]
 [3.6515500e-02 2.3412127e+00 1.0910000e+00 1.1840000e+00]
 [4.3920800e-02 3.0925140e+00 9.0207500e-01 1.1810000e+00]
 [4.9943600e-02 3.7521000e+00 5.4279700e-01 1.1800000e+00]]
```

Objectif #6 : utiliser l'entrée et la sortie standard, écrire et lire des fichiers avec un format défini

☆☆ Sauvegarder un tableau Numpy dans un fichier de données :

```
np.savetxt ('exoplanets-SI.dat', np.c_ [a_SI, p_SI, mp_SI, me_SI ], fmt = '%10.3e')
```

```
# change units to SI
a_SI = a * au2m
p_SI = p * day2seconds
mp_SI = mp * M_jupiter
me_SI = me * M_sun
```

```
2.394e+11  5.807e+07  2.124e+28  2.387e+30
1.139e+10  6.055e+05  1.428e+25  2.387e+30
7.774e+10  1.079e+07  4.826e+25  2.387e+30
1.101e+11  1.820e+07  4.129e+26  2.387e+30
7.936e+09  3.518e+05  2.521e+27  2.387e+30
1.873e+11  4.075e+07  1.352e+28  2.327e+30
1.491e+11  2.865e+07  1.779e+27  2.387e+30
3.491e+11  1.026e+08  2.217e+28  2.369e+30
6.957e+09  2.899e+05  1.302e+27  2.369e+30
1.691e+10  1.099e+06  1.706e+26  2.367e+30
5.159e+09  1.852e+05  1.868e+27  2.367e+30
1.664e+11  3.392e+07  2.402e+27  2.367e+30
1.051e+10  5.390e+05  4.243e+25  2.367e+30
6.805e+09  2.805e+05  1.938e+27  2.367e+30
3.897e+11  1.214e+08  8.986e+27  2.367e+30
7.470e+09  3.227e+05  9.652e+26  2.367e+30
7.159e+09  3.032e+05  8.732e+26  2.361e+30
5.463e+09  2.023e+05  2.071e+27  2.355e+30
6.570e+09  2.672e+05  1.712e+27  2.349e+30
7.471e+09  3.242e+05  1.030e+27  2.347e+30
7.018e+09  2.951e+05  5.620e+26  2.347e+30
```

exoplanets-SI.dat

Objectif #6 : utiliser l'entrée et la sortie standard, écrire et lire des fichiers avec un format défini

☆☆ Sauvegarder un tableau Numpy dans un fichier de données :

```
np. … ('exoplanets-SI.dat', np. … [a_SI, p_SI, mp_SI, me_SI ], fmt = '%10.3e')
```

☆ Format des données : `fmt = '%10.3e'`

`10` = nombre total de caractères (signes compris)

`3` = nombre de chiffres après le point décimal

`e` = type d'affichage exponentiel

```
# change units to SI
a_SI = a * au2m
p_SI = p * day2seconds
mp_SI = mp * M_jupiter
me_SI = me * M_sun
```

```
2.394e+11  5.807e+07  2.124e+28  2.387e+30
1.139e+10  6.055e+05  1.428e+25  2.387e+30
7.774e+10  1.079e+07  4.826e+25  2.387e+30
1.101e+11  1.820e+07  4.129e+26  2.387e+30
7.936e+09  3.518e+05  2.521e+27  2.387e+30
1.873e+11  4.075e+07  1.352e+28  2.327e+30
1.491e+11  2.865e+07  1.779e+27  2.387e+30
3.491e+11  1.026e+08  2.217e+28  2.369e+30
6.957e+09  2.899e+05  1.302e+27  2.369e+30
1.691e+10  1.099e+06  1.706e+26  2.367e+30
5.159e+09  1.852e+05  1.868e+27  2.367e+30
1.664e+11  3.392e+07  2.402e+27  2.367e+30
1.051e+10  5.390e+05  4.243e+25  2.367e+30
6.805e+09  2.805e+05  1.938e+27  2.367e+30
3.897e+11  1.214e+08  8.986e+27  2.367e+30
7.470e+09  3.227e+05  9.652e+26  2.367e+30
7.159e+09  3.032e+05  8.732e+26  2.361e+30
5.463e+09  2.023e+05  2.071e+27  2.355e+30
6.570e+09  2.672e+05  1.712e+27  2.349e+30
7.471e+09  3.242e+05  1.030e+27  2.347e+30
7.018e+09  2.951e+05  5.620e+26  2.347e+30
```

exoplanets-SI.dat

Pour la prochaine séance (lundi 15/02, salle 102) :

- Déposer, sur moodle, le mini-projet avant **dimanche 06/07 21h00.**
- Faire les calepins **itérations** et **fonctions** de la séquence 3