Modélisation numérique en physique

Séquence 3 : Python intermédiaire (II)

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.*

\+ 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
```

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.

☆☆ 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()
```

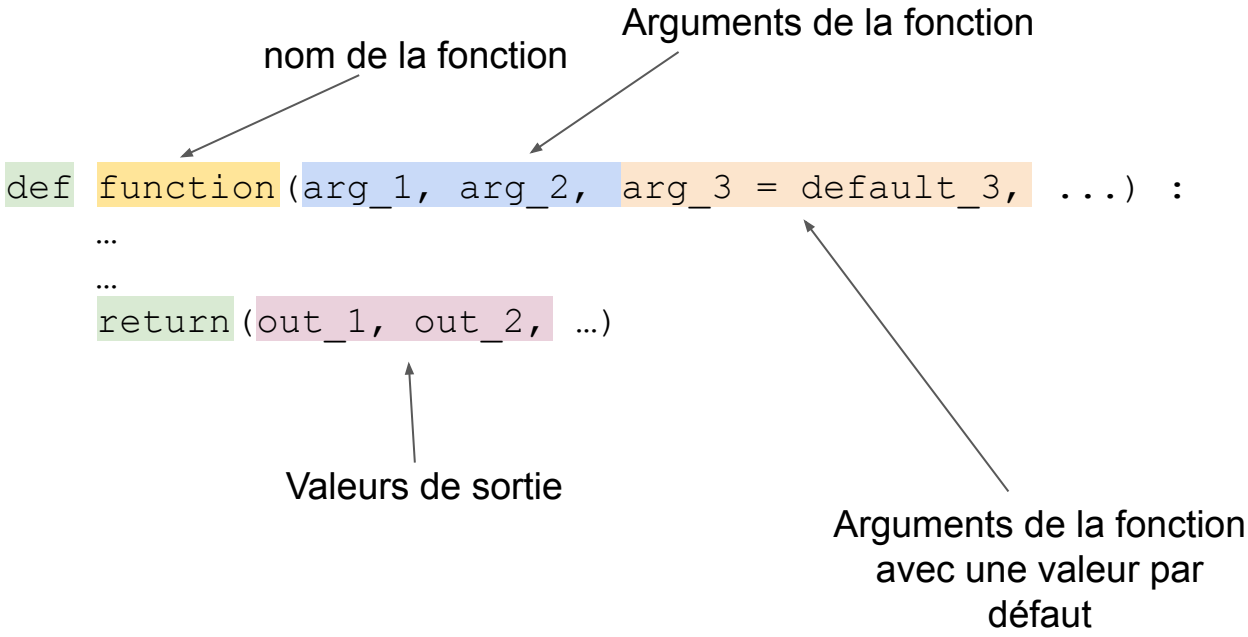
```
Plante : gentiane
1    7.5
2    7.8
3    8.3
4    8.4
Name: gentiane, dtype: float64

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, ...)
```

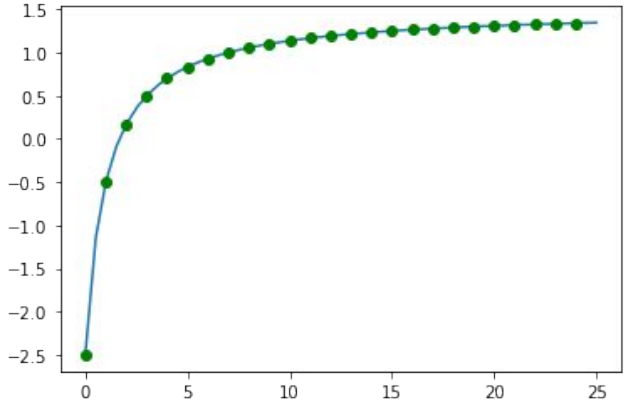
Autant de variables que de valeurs en sortie

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

Objectif #6 : Représentation graphique d'une série mathématique

Définir la fonction suite	$F(n) = \frac{3n - 5}{2n + 2}$	`def F(n):`
		`    ''' retourne l'element Sn de la suite '''`
		`    return (3*n - 5) / (2*n + 2)`

Calculer les éléments de ma suite et les stocker dans un tableau	$S_n = \frac{3n - 5}{2n + 2}$	`S = np.array([])`
		`n = list(range(N))`
		
		`for n in range(N) :`
		`    S = np.append(S, F(n))`



Objectif #6 : Représentation graphique d'une série mathématique

Représenter une suite définie par récurrence :

$$u_{n+1} = u_n - \ln(u_n)$$

Initialiser la suite	$u_0 = 10$	
Définir la fonction associée à la suite	$G(x) = x - \ln(x)$	`def G(x):` `    return x - np.log(x)`

Calculer les termes suivants	(voir tableau)	`for k in range(N):`
		`    # etape 2: image du point U0 avec une ligne`
		`    #   verticale -> U_01`
		`    u1 = G(u0)`
		`    plt.plot([u0, u0], [u0, u1], 'k')`
		`    plt.plot([u0], [u1], 'og')`
		`    # etape 3: placer le point U1 sur la bissectrice`
		`    #   avec une ligne horizontale depuis U_01`
		`    plt.plot([u0, u1], [u1, u1], 'k')`
		`    plt.plot([u1], [u1], 'og')`
		`    # recurrence:`
		`    u0 = u1`

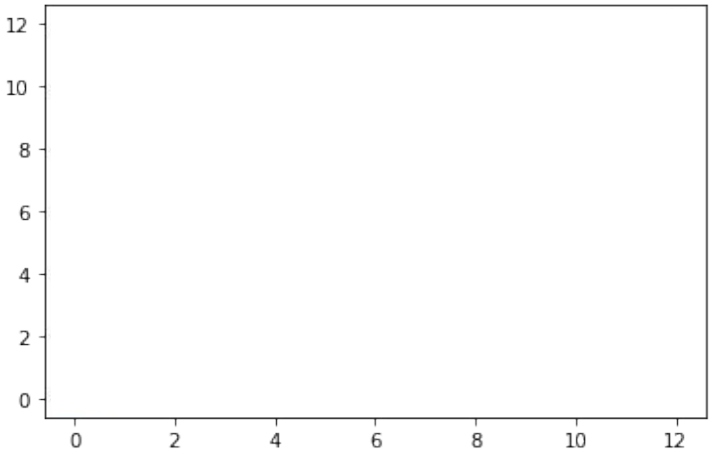
Objectif #6 : Représentation graphique d'une série mathématique

Représenter une suite définie par récurrence :

```
# Initialisation
plt.plot([u0], [u0], 'og')

# Récurrence
for k in range(N):
    # etape 2: image du point U0 avec une ligne
    #   verticale -> U_01
    u1 = G(u0)
    plt.plot([u0, u0], [u0, u1], 'k')
    plt.plot([u0], [u1], 'og')
    # etape 3: placer le point U1 sur la bissectrice
    #   avec une ligne horizontale depuis U_01
    plt.plot([u0, u1], [u1, u1], 'k')
    plt.plot([u1], [u1], 'og')
    # recurrence:
    u0 = u1
```

$$u_{n+1} = u_n - \ln(u_n)$$



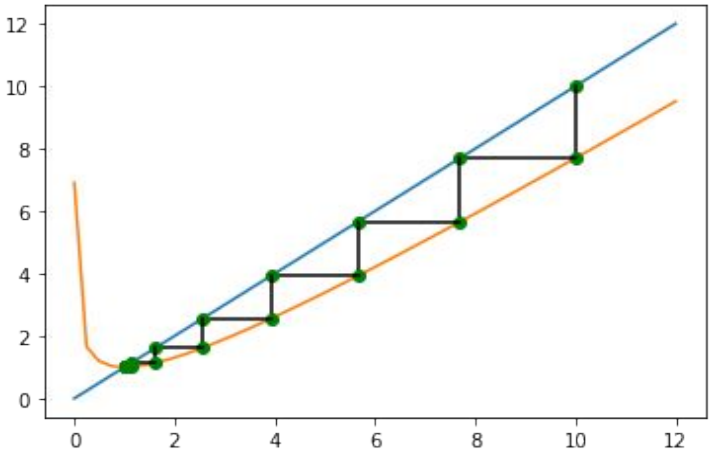
Objectif #6 : Représentation graphique d'une série mathématique

Représenter une suite définie par récurrence :

```
# Initialisation
plt.plot([u0], [u0], 'og')

# Récurrence
for k in range(N):
    # etape 2: image du point U0 avec une ligne
    #   verticale -> U_01
    u1 = G(u0)
    plt.plot([u0, u0], [u0, u1], 'k')
    plt.plot([u0], [u1], 'og')
    # etape 3: placer le point U1 sur la bissectrice
    #   avec une ligne horizontale depuis U_01
    plt.plot([u0, u1], [u1, u1], 'k')
    plt.plot([u1], [u1], 'og')
    # recurrence:
    u0 = u1
```

$$u_{n+1} = u_n - \ln(u_n)$$



Fiche : Formatage des affichages

```
a_value = 12
b_value = 13.4563750
```

Procédure usuelle :

```
print("La valeur de a est", a_value, "et celle de b est", b_value ,".")
```

```
La valeur de a est 12 et celle de b est 13.456375.
```

```
print("La valeur de a est " + str(a_value) + "et celle de b est " + str(b_value) + ".")
```

```
La valeur de a est 12 et celle de b est 13.456375.
```

Source : https://python.sdv.univ-paris-diderot.fr/03_affichage/

Fiche : Formatage des affichages

```
a_value = 12
b_value = 13.4563750
```

L'écriture formatée :

```
print(f"La valeur de a est {a_value} et celle de b est {b_value}.")
```

```
La valeur de a est 12 et celle de b est 13.456375.
```

```
print(f"La valeur de a est {a_value} et celle de b est {b_value:.2f}.")
```

```
La valeur de a est 12 et celle de b est 13.46.
```

(Voir le calepin etu-04-ajustement-modele/etu-01-entrees-sorties pour les formats définis)

Source : https://python.sdv.univ-paris-diderot.fr/03_affichage/

Fiche : Formatage des affichages

```
a_value = 12
b_value = 13.4563750
```

L'opérateur % :

```
print("La valeur de a est %d et celle de b est %.2f." % (a_value, b_value))
```

```
La valeur de a est 12 et celle de b est 13.46.
```

La méthode .format() :

```
print("La valeur de a est {} et celle de b est {:.4f}.".format(a_value, b_value))
```

```
La valeur de a est 12 et celle de b est 13.4564.
```

(Voir le calepin etu-04-ajustement-modele/etu-01-entrees-sorties pour les formats définis)

Source : https://python.sdv.univ-paris-diderot.fr/03_affichage/

- *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.*

Pour la prochaine séance (lundi 14/02) :

- Déposer, sur moodle, le mini-projet avant **dimanche 13/02 21h00.**
- Faire le calepin **Entrées-sorties**

Date limite pour les seconds rendus du MP1 :

- **Dimanche 20/02 à 21h**