IngeSUP - TD 08 - Les fonctions 2

> *"Certains hommes changent de parti en fonction de leurs opinions, d'autres changent d'opinion en fonction de leur parti."*

Winston Churchill

Exercice 08.1 : Listes et fonctions

1. Réalisez une fonction nommée `Permuter()` qui permute deux valeurs d'un tableau `T` passé en paramètre. Les indices `i` et `j` des éléments à permuter doivent aussi être en paramètres. La fonction retournera le tableau résultant après la permutation.

Entrée [1]:

```
def Permuter(T,i,j):
    copie=T[j]
    T[j]=T[i]
    T[i]=copie
    return T
```

2. **A l'extérieur de toute fonction** créez un tableau `tab` = [2,1,3] , puis appelez votre fonction avec le tableau `tab` , `i` = 0 et `j` = 1. Affichez le résultat ainsi obtenu.

Entrée [3]:

```
tab=[2,1,3]
print(Permuter(tab,0,1))
```

```
[1, 2, 3]
[1, 2, 3]
```

3. Le tableau `T` a t-il été modifié par son appel dans la fonction `Permuter()` ? Pourquoi ?

Réponse:

Exercice 08.2 : Calcul des termes d'une suite définie par $u_n = f(n)$

On considère la suite (U_n) définie pour tout entier naturel n par

$$u_n = \frac{3n-9}{2n+1}$$

Créez une fonction `U()` qui prend en entrée un entier naturel `n` et renvoie en sortie la valeur de u_n.

Entrée [5]:

```
def U(n):
    return (3*n-9)/(2*n+1)

print(U(0))
print(U(10))
```

```
-9.0
1.0
```

Exercice 08.3 : Calcul des termes d'une suite définie par récurrence

Pour calculer à l'aide d'un programme les termes d'une suite définie par récurrence, l'idée est tout simplement de calculer au fur et à mesure les valeurs de la suite en les sauvegardant dans une seule variable u qui commence à u_0.

Dans la fenêtre ci-dessous, on a déjà commencé à écrire un programme pour calculer la valeur de u_n définie par $u_0 = 5$ et $u_{n+1} = 2u_n - 3$.

Remplacez les **...** par ce qu'il faut pour que le programme fonctionne.

Entrée [6]:

```
# Remplacez ci-dessous les ... par le code qu'il faut
# N'oubliez pas : testez votre suite sur quelques termes et vérifiez à la main...
# Attention à l'indentation !

def suite_exo3(n):
    u0 = 5
    if n==0:
        return u0
    else:
        for i in range(1, n+1):
            u0 = 2*u0-3
    return u0

suite_exo3(6)
```

Out[6]:

```
131
```

Exercice 08.4 : Calcul des termes d'une suite définie par récurrence II

Dans cet exercice, on considère une suite u définie par $u_{n+1} = 3 - 4u_n$ et de premier terme u_0 (non indiqué).

Le but de cet exercice est de créer un programme qui prend en entrée les valeurs de `n` et aussi `u_0` et affiche la valeur de `u_n`.

> Entrée : Un entier `n` et `u_0`.

> Sortie : La valeur de `u_n`.

Entrée [1]:

```
def suite_exo4(n, u_0):
    if n==0:
        return u_0
    else:
        u_n = u_0
        for i in range(1 , n+1):
            u_n = 3-4*u_n
        return u_n

print(suite_exo4(2,1))
```

```
7
```

Exercice 08.5 : Calcul approché de π

La formule de Leibniz est une série alternée calculant une approximation de π :

$$4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}=\frac{4}{1}-\frac{4}{3}+\frac{4}{5}-\frac{4}{7}+\frac{4}{9}-\frac{4}{11}+\cdots$$

Cette série converge si lentement que près de 200 termes sont nécessaires pour calculer π avec deux décimales exactes !

1. Écrivez une fonction permettant de calculer une approximation de π en fonction du nombre de termes n souhaité (n est un paramètre de la fonction).

Entrée [2]:

```
def Leibniz(n):
    somme=0
    for k in range(n):
        u_k=(-1)**k/(2*k+1)
        somme+=u_k
    return somme*4

print(Leibniz(50000))
```

```
3.1415726535897814
```

2. A l'aide de la fonction précédente, écrivez une fonction calculant l'approximation de π, de telle sorte que le calcul s'arrête lorsque la différence entre la série de $K+1$ termes et la série de K termes soit inférieure à 0,001.

Indication: Pour calculer la différence entre la série de $K+1$ termes et la série de K termes sans tenir compte des variations de signe vous aurez peut-être besoin de la fonction valeur absolue, incarnée en python par `abs()` .

Entrée [14]:

```
def Leibniz():
    somme=0
    k=0
    while abs((-1)**k/(2*k+1)) > 0.001:
        u_k=(-1)**k/(2*k+1)
        somme+=u_k
        k+=1
    return somme*4

print(Leibniz())
```

```
3.139592655589785
```

Exercice 08.6 : Calcul approché de racines carrées

On considère la suite définie par récurrence de la façon suivante :

$$\begin{cases} x_0 = 2 \\ \forall n \in \mathbb{N}, x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n}) \end{cases}$$

1. Programmez cette suite numérique. Votre fonction prendra deux paramètres : `n` qui correspond au terme de la suite dont on veut la valeur et `a` une constante précisée par l'utilisateur lors de l'appel à la fonction.

Entrée [16]:

```
def U(n,a):
    u0 = 2
    if n==0:
        return u0
    else:
        for i in range(1, n+1):
            u0 = (1/2)*(u0+a/u0)
        return u0
```

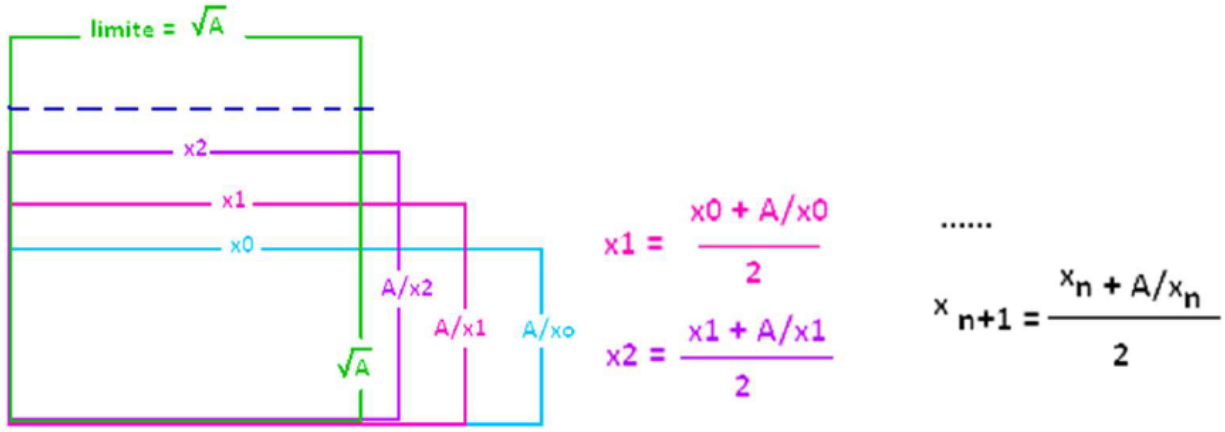
2. Affichez les valeurs des quatre premiers termes de la suite qui correspond à `a = 1`.

Entrée [17]:

```
print(U(0,1))
print(U(1,1))
print(U(2,1))
print(U(3,1))
```

```
2
1.25
1.025
1.0003048780487804
```

La méthode de Héron permet d'évaluer la racine carrée d'un réel A en déformant progressivement un rectangle ayant une aire égale à A :



3. En modifiant la suite précédente telle que $x_0 = 1$ et $a = 2$ on peut obtenir une bonne approximation de $\sqrt{2}$ en se reposant sur la **méthode de Héron**. Programmez cette nouvelle suite.

Entrée [27]:

```
def racineDe2(n):
    u0 = 1
    if n==0:
        return u0
    else:
        for i in range(1, n+1):
            u0 = (1/2)*(u0+2/u0)
    return u0

print(racineDe2(50))
```

```
1.414213562373095
```

4. A partir de quelle terme de la suite a t-on $|x_{k+1} - x_k| \leq 0,001$?

Entrée [22]:

```
def limite():
    u0 = 1
    k=0
    while abs((1/2)*(u0+2/u0) - u0)>0.001:
        u0 = (1/2)*(u0+2/u0)
        k+=1
    return k

print(limite())
```

```
3
```

Exercice 08.7 : Calcul des termes d'une suite définie par récurrence double

Écrivez une fonction `suite_exo7()` qui calcule le n-ème terme de la suite définie par :

$$\begin{cases} U_1 = U_2 = 1 \\ \forall n \geq 3, U_{n+1} = 3U_{n-1} + 2U_{n-2} \end{cases}$$

Entrée [32]:

```
def suite_exo7(n):
    a0 = a1 = 1
    if n == 0:
        return a0
    elif n == 1:
        return a1
    else:
        for i in range(2, n+1):
            a = 3*a1 + 2*a0
            a0 = a1
            a1 = a
        return a1
```

Exercice 08.8 : Somme arithmétique

Écrivez une fonction `somme_arithmetique()` qui calcule la valeur de la somme $1^p + 2^p + 3^p + \cdots + n^p$ pour des valeurs de `n` et p données en paramètre.

Entrée [33]:

```
def somme_arithmétique(n,p):
    somme=0
    for i in range(n+1):
        somme+=i**p
    return somme

print(somme_arithmétique(10,3))
```

```
3025
```

Exercice 08.9 : Calcul de moyenne

Écrivez une fonction `moyenne()` qui calcule la moyenne des valeurs d'un tableau `Tab` donné en paramètre.

Entrée [28]:

```
def moyenne(tab):
    somme=sum(tab)
    return somme/len(tab)

print(moyenne([1,2,3,4]))
```

```
2.5
```

Exercice 08.10 : Calcul de variance

A l'aide de la fonction `moyenne()` , écrivez une fonction `variance()` qui calcule la variance des valeurs d'un tableau `T` donné en paramètre.

Indication: Pour rappel, le calcul de la variance ***V*** de ***n*** valeurs s'effectue avec la ***moyenne m*** de la manière suivante :

$$V(x) = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - m)^2$$

Ici les x_i représentent les valeurs du tableau.

Entrée [31]:

```
def variance(T):
    m=moyenne(T)
    somme=0
    for i in T:
        somme+=(i-m)**2
    return somme/len(T)

print(variance([0,2,3,5]))
```

```
3.25
```

Exercice 08.11 : Fonctions, listes et fichiers

1. Écrivez une fonction `getNotes()` qui récupère les notes d'un fichier texte dont le nom est passé en paramètres. La fonction `getNotes()` admet un deuxième paramètre qui correspond au mode d'ouverture du fichier. Ce paramètre est par défaut fixé à *"r"* (pour ouvrir en mode lecture).

Les notes sont stockées dans un tableau (sous forme de nombres réels) et le tableau est renvoyé par la fonction.

Entrée [ ]:

```
def getNotes(nomFichier, mode="r"):
    liste_finale = []
    f = open(nomFichier, "r")
    liste = f.readlines()    # Les notes sont dans un tableau...
    #print(liste)    # Mais le format de ces notes (nombre suivi du '\n') oblige à un traitement supplémentaire
    n = len(liste)

    for i in range(n):
        liste_finale.append(float(liste[i]))   # "float" enlève les '\n' et transforme le caractère en réel

    f.close()

    return liste_finale
```

2. Testez la fonction `getNotes()` sur un fichier où vous aurez enregistré des notes à la main. Il y'a une note par ligne.

Entrée [ ]:

```
print(getNotes("notes.txt", "r"))
```

3. Est-il possible d'appeler la fonction `getNotes()` en mettant les arguments dans le désordre ? Est-il possible de l'appeler en ne spécifiant qu'un seul argument ? Si oui, comment ?

Il n'est pas possible d'appeler `getNotes` en mettant les arguments dans le désordre car il y'a deux arguments. Le premier argument est obligatoire et correspond au nom du fichier. Le deuxième est optionnel et correspond au mode.

Or, en Python, les arguments obligatoires doivent toujours être mis en premier.

Cependant, il est possible d'appeler la fonction en ne spécifiant qu'un seul argument (le nom), vu que l'autre argument (le mode) est optionnel. Voici la méthode :

```
getNotes("notes.txt")
```

OU

```
print(getNotes("notes.txt"))    # si on  veut voir quelque chose s'afficher à l'écran
```

Ecrivez votre réponse ici

4. A l'aide de la fonction `getNotes()` et de la fonction `moyenne()` de l'exercice *07.9* , écrivez une fonction `moyenne_fichier()` qui calcule la moyenne des notes d'un fichier passé en paramètre.

Entrée [ ]:

```
def moyenne_fichier(nomFichier):
    tableau_notes = getNotes(nomFichier)
    moyenne_notes = moyenne(tableau_notes)
    return moyenne_notes

print(moyenne_fichier("notes.txt"))
```

5. A l'aide de la fonction `getNotes()` et de la fonction `variance()` de l'exercice *07.10* , écrivez une fonction `variance_fichier()` qui calcule la variance des notes d'un fichier passé en paramètres.

Entrée [ ]:

```
def variance_fichier(nomFichier):
    tableau_notes = getNotes(nomFichier)
    variance_notes = variance(tableau_notes)
    return variance_notes

print(variance_fichier("notes.txt"))
```

6. Écrivez une fonction `is_ordered_asc()` qui détermine si un fichier donné en paramètres contient des notes triées en ordre croissant.

Indication: Récupérez les notes dans un tableau et regardez si le tableau est trié en ordre croissant...

Entrée [ ]:

```
def is_ordered_asc(nomFichier):
    tableau_notes = getNotes(nomFichier)

    # Maintenant que toutes les notes sont dans un tableau
    # On vérifie que chaque note placée dans une case est plus petite
    # que la note qui est dans la case juste à sa droite

    n = len(tableau_notes)

    # On s'arrête à n-1 car à sa droite il y'a n et n
    # est la dernière case du tableau, après il n'y a
    # plus rien

    for j in range(n-1):
        if tableau_notes[j] > tableau_notes[j+1]:
            return False
    return True

print(is_ordered_asc("notes.txt"))
```

7. Écrivez une fonction `is_ordered_desc()` qui détermine si un fichier donné en paramètres contient des notes triées en ordre décroissant.

Indication: Récupérez les notes dans un tableau et regardez si le tableau est trié en ordre décroissant...

Entrée [ ]:

```
def is_ordered_desc(nomFichier):
    tableau_notes = getNotes(nomFichier)

    # Maintenant que toutes les notes sont dans un tableau
    # On vérifie que chaque note placée dans une case est plus grande
    # que la note qui est dans la case juste à sa droite

    n = len(tableau_notes)

    # On s'arrête à n-1 car à sa droite il y'a n et n
    # est la dernière case du tableau, après il n'y a
    # plus rien

    for j in range(n-1):
        if tableau_notes[j] < tableau_notes[j+1]:
            return False
    return True

print(is_ordered_desc("notes.txt"))
```

8. En déduire une fonction `is_ordered()` qui détermine si un fichier, donné en paramètres, contient des notes triées en ordre croissant ou en ordre décroissant selon une valeur booléenne, par exemple :

```
is_ordered(nomFichier, croissant=True)    # Pour vérifier si c'est trié en ordre croissant
is_ordered(nomFichier, croissant=False)    # Pour vérifier si c'est trié en ordre décroissant
```

indication: Vous pouvez vous servir des deux fonctions précédentes.

Entrée [ ]:

```
def is_ordered(nomFichier, croissant):
    if croissant == True:
        valeur = is_ordered_asc(nomFichier)
    else:
        valeur = is_ordered_desc(nomFichier)
    return valeur

print(is_ordered("notes.txt", croissant=True))
print(is_ordered("notes.txt", croissant=False))
```

Corrigé du TD 08

Vous pouvez retrouver le corrigé de ce TD ici (Corrig%C3%A9s/Corrig%C3%A9_TD%2008.ipynb).