UVSQ ISTY Institut des Sciences et Techniques des Yvelines – université PARIS-SACLAY – CAMPUS DE MANTES EN YVELINES – CAMPUS DE SAINT-QUENTIN-EN-YVELINES	***LES SUITES RÉCURRENTES***
	Python

I. Introduction

En mathématiques, une suite, notée habituellement **(u_n)**, est composée d'une succession de nombres ou de termes ordonnés : **u_0, u_1, ..., u_n, ...**

Les suites sont présentes dans de nombreux domaines. En économie, elles peuvent par exemple servir à connaître la valeur d'un placement ou d'un capital au bout de **n** années.

On cherche généralement à calculer le terme de rang **n** à l'aide d'une fonction ou d'une relation de récurrence.

Dans notre cas, on va s'intéresser aux suites récurrentes dans lesquelles chaque terme s'obtient à partir des précédents à l'aide d'une relation de récurrence :

1. Suite arithmétique
2. Suite géométrique
3. Suite arithmético-géométrique
4. Suite récurrente linéaire

Puis, on va montrer comment générer en Python les premiers termes de ces suites, en donnant à chaque fois un exemple de mise en œuvre.

Enfin, on écrira une fonction générique qui prendra comme argument la fonction de récurrente de son choix.

II. Suites récurrentes

En mathématiques, une suite récurrente est une suite associée à une fonction f (d'une ou plusieurs variables) appelée fonction de récurrence, laquelle permet de calculer chaque terme à partir des précédents par une relation de récurrence de la forme :

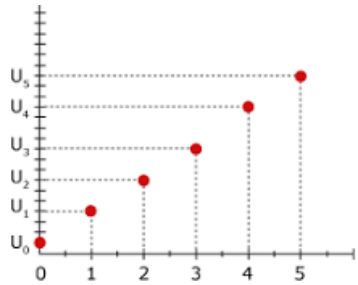
$u_{n+p}=f(u_n,u_{n+1},\dots,u_{n+p-1})$ où n est un entier naturel et p un entier supérieur à **0**.

Soit plus simplement pour une fonction à une seule variable :

$u_{n+1}=f(u_n)$

II-A. Suite arithmétique

En mathématiques, une suite arithmétique est une suite dans laquelle chaque terme permet de déduire le suivant en lui ajoutant une constante appelée **raison**.



Prenons par exemple la liste de termes :

u_0 = 1

u_1 = 3

u_2 = 5

u_3 = 7

...

On ajoute donc **2** au terme d'indice n pour obtenir le suivant d'indice n**+1**. La relation de récurrence de la suite peut alors s'écrire :

$u_{n+1}=u_n+2$ avec comme terme initial : $u_0 = 1$

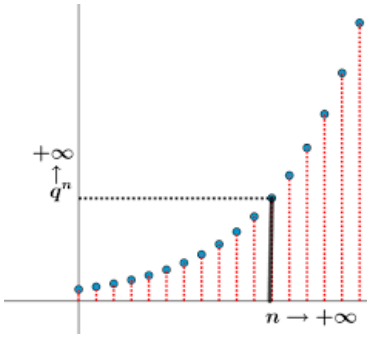
Il s'agit en fait d'une suite arithmétique de **raison 2** et de **premier terme 1**.

Dans le cas général, une suite arithmétique est donc définie par la relation de récurrence :

$u_{n+1} = u_n + r$ avec le terme initial : $u_0 = a$ où la constante r désigne la raison de la suite et n est un entier naturel.

II-B. Suite géométrique

Une suite géométrique est une suite de nombres dans laquelle chaque terme permet de déduire le suivant en le multipliant par un facteur constant appelé **raison**.



Prenons maintenant la liste ordonnée de termes :

$u_0 = 5$

$u_1 = 10$

$u_2 = 20$

$u_3 = 40$

$u_4 = 80$

On multiplie bien chaque terme par **2** pour obtenir le suivant. La relation de récurrence de la suite peut alors s'écrire :

$u_{n+1}=2.u_n$ avec $u_0 = 5$ comme terme initial

Dans le cas général, une suite géométrique est donc définie par la relation de récurrence :

$u_{n+1} = q.u_n$ avec le terme initial : $u_0 = a$ où la constante q désigne la raison de la suite et n est un entier naturel.

II-C. Suite arithmético-géométrique

Une suite arithmético-géométrique est une suite satisfaisant une relation de récurrence affine, généralisant ainsi les définitions des suites arithmétiques et géométriques.

Elle est définie par la relation de récurrence :

$u_{n+1}=a.u_n+b$ avec le terme initial u_0 **et ;** a et b étant des constantes et n un entier naturel

II-D. Suite récurrente linéaire

Une suite récurrente linéaire d'ordre **p** est définie par la relation de récurrence :

$u_{n+p}=a_0.u_n+a_1.u_{n+1}+...+a_{p-1}.u_{n+p-1}$

$a_0, a_1, .., a_{p-1}$ étant des constantes. Les suites récurrentes linéaires d'ordre **1** sont les suites géométriques.

III. Implémentation en Python

On souhaite maintenant traduire les différentes suites récurrentes en Python, en cherchant bien sûr à écrire du code le plus lisible possible.

On donnera également quelques exemples de mise en œuvre en mathématiques financières élémentaires.

III-A. Suite arithmétique

III-A-1. Fonction Python

En partant de la relation de récurrence : $u_{n+1} = u_n + r$ avec le terme initial u_0, on peut facilement écrire la fonction Python :

```
def suite_arithmetique(u0,r,n):
    # génère les n+1 premiers termes de la suite arithmétique de raison r et de
terme initial u0

    # on initialise la variable ui et la liste u avec le terme initial de la suite
    ui = u0; u = [u0]

    # parcours des indices des termes de la suite : 1 -> n
    for i in range(1,n+1):
        ui += r # ajout de r au terme précédent pour obtenir le terme d'indice i

        u.append(ui) # ajout du terme obtenu à la liste

    # renvoie la liste des n+1 premiers termes de la suite arithmétique
    return u
```

Elle renvoie les n**+1** premiers termes de la suite.

Note : au lieu d'ajouter chaque terme à une liste, on peut également choisir de l'afficher directement dans la boucle :

```
...

    # parcours des indices des termes de la suite : 1 -> n
    for i in range(1,n+1):
        ui += r # ajout de r au terme précédent pour obtenir le terme d'indice i

        print(f'u({i}) = {ui}') # affiche u(i) = ..

    ...
```

III-A-2. Évolution d'un capital à intérêts simples

Une personne place un capital de C_0**=5000** € à intérêts simples au taux annuel de **t=0.05**, c'est à dire qu'à la fin de chaque année on lui donne un intérêt égal à **5%** de la somme déposée initialement.
Naturellement cette personne désire connaître la somme dont elle disposera au bout de **1** an, **2** ans, ...
On définit donc la suite de termes :
$C_0, C_1, ..., C_n, ...$

En partant d'un capital initial C_0**=5000**, le capital augmente donc de **250** euros par an :
$C_{n+1} = C_n + 250$
Il suit donc une progression arithmétique de raison **r=250** et de terme initial C_0**=5000**.
La fonction précédente nous permet ainsi d'obtenir la liste des **11** premières valeurs du capital représentant son évolution sur **10** années :

```
#1er terme de la suite : capital initial
c0 = 5000
```

```
# raison de la suite : augmentation annuelle du capital de 250 euros
r = 250

# liste des valeurs successives du capital sur 10 années
valeurs_capital = suite_arithmetique(c0,r,10)

print("Évolution du capital sur 10 années :")
print(valeurs_capital)
```

Le code affiche :
Évolution du capital sur 10 années :
[5000, 5250, 5500, 5750, 6000, 6250, 6500, 6750, 7000, 7250, 7500]

II-B. Progression géométrique

III-B-1. Fonction Python

En partant cette fois de la relation de récurrence : $u_{n+1} = q.u_n$ avec le terme initial u_0, on peut facilement obtenir la fonction Python :

```
def suite_geometrique(u0,q,n):
    # génère les n+1 premiers termes de la progression géométrique de raison q et de
terme initial u0

    # on initialise la variable ui et la liste u avec le terme initial de la suite
    ui = u0; u = [u0]

    # parcours des indices des termes de la suite : 1 -> n
    for i in range(1,n+1):
        ui *= q # multiplication du terme précédent par q pour obtenir le terme
d'indice i
        u.append(ui) # ajout du terme obtenu à la liste

    # renvoie la liste des n+1 premiers termes de la suite géométrique
    return u
```

Elle renvoie donc les n**+1** premiers termes de la suite géométrique.

III-B-2. Évolution d'un capital à intérêts composés

Une personne place un capital de C_0**=5000** € à intérêts composés au taux annuel de **t=0.05**, c'est à dire qu'à la fin de chaque année on lui donne un intérêt égal à **5%** du capital précédent.
On obtient ainsi la somme C_{n+1} au bout de n**+1** années en ajoutant **0.05**$\times C_n$ au capital précédent C_n :
$C_{n+1} = C_n + 0.05 \times C_n$
$C_{n+1} = 1.05 \times C_n$ avec comme terme initial : $C_0 = 5000$
L'évolution du capital suit donc une progression géométrique de raison **q=1.05** et de terme initial $C_0 = 5000$.
La fonction précédente nous permet ainsi d'obtenir l'évolution du capital sur par exemple **10** années :

```
#1er terme de la suite : capital initial
c0 = 5000

# raison de la suite géométrique : q = (1+t) avec t le taux d'intéret annuel
q = 1.05

# liste des valeurs successives du capital sur 10 années
valeurs_capital = suite_geometrique(c0,q, 10)

# liste des valeurs du capital arrondies à 2 décimales
valeurs_capital2 = [float('%.2f' % ci) for ci in valeurs_capital]

print("Évolution du capital sur 10 années :")
```

```
print(valeurs_capital2)
```

Le code affiche :
Évolution du capital sur 10 années :
[5000.0, 5250.0, 5512.5, 5788.12, 6077.53, 6381.41, 6700.48, 7035.5, 7387.28, 7756.64, 8144.47]

III-C. Progression arithmético-géométrique

III-C-1. Fonction Python

En partant de la relation de récurrence :
$u_{n+1} = a.u_n + b$ avec le terme initial u_0
On peut facilement écrire la fonction Python :

```
def suite_arithmetico_geometrique(u0,a,b,n):
    # génère les n+1 premiers termes de la progression arithmético-géométrique de paramètres a et b, et de terme initial u0

    # on initialise la variable ui et la liste u avec le terme initial de la suite
    ui = u0; u = [u0]

    # parcours des indices des termes de la suite : 1 -> n
    for i in range(1,n+1):
        ui = a*ui + b # calcul du terme d'indice i à partir du terme précédent
        u.append(ui) # ajout du terme obtenu à la liste

    # renvoie la liste des n+1 premiers termes de la suite arithmético-géométrique
    return u
```

III-C-2. Évolution du capital restant dû

Un capital **c** emprunté à un taux mensuel **t** et remboursé par mensualités constantes **m** conduit à la construction d'une suite arithmético-géométrique.
Si R_n représente le capital restant dû au bout de n mensualités, la suite (R_n), est définie par la relation de récurrence :
$R_{n+1} = (1+t)R_n - m$
Pour un capital de départ **c=10000**, un taux mensuel **t=0.004** et un remboursement mensuel de **m=1000**, on aboutit donc à la formule de récurrence :
$R_{n+1} = 1.004 \times R_n - 1000$
La fonction Python nous permet ainsi d'obtenir l'évolution du capital restant dû sur par exemple **10** mois :

```
#1er terme de la suite : capital initial
r0 = 10000

# raison de la suite géométrique : q = (1+t) avec t le taux d'intérêt mensuel
a = 1.004
b = -1000

# liste des valeurs successives du capital restant dû sur 10 mois
valeurs_capital = suite_arithmetico_geometrique(r0,a, b, 10)

# liste des valeurs du capital arrondies à 2 décimales
valeurs_capital2 = [float('%.2f' % ci) for ci in valeurs_capital]

print("Évolution du capital restant dû sur 10 mois :")
print(valeurs_capital2)
```

Le code affiche :
Évolution du capital restant dû sur 10 mois :
[10000.0, 9040.0, 8076.16, 7108.46, 6136.9, 5161.45, 4182.09, 3198.82, 2211.62, 1220.46, 225.34]

On peut également chercher à connaître le nombre de mois nécessaires pour rembourser en totalité l'emprunt, c'est à dire quand le capital restant dû est égal à **0**.

III-D. Suite récurrente linéaire

III-D-1. Fonction Python

A partir de la relation de récurrence :

$u_{n+p} = a_0.u_n + a_1.u_{n+1} + ... + a_{p-1}.U_{n+p-1}$: a_0, a_1, .., a_{p-1} étant des constantes

On obtient la fonction Python :

```
def suite_recurrente_lineaire(a,u,n):
    # génère les n+1 premiers termes de la suite récurrente linéaire à partir des listes de valeurs initiales a et u

    # nombre de constantes ai ou de termes dans la relation de récurrence
    p=len(a)

    # parcours des indices des termes de la suite : p -> n
    for i in range(p,n+1):
        # calcul du terme d'indice i
        ui = 0
        for aj,uj in zip(a,u[i-p:]):
            ui += aj*uj

        u.append(ui)

    # renvoie la liste des n+1 premiers termes de la suite récurrente linéaire
    return u
```

Les arguments a et **u** désignent les listes contenant les constantes et les termes initiaux de la suite.

III-D-2. Générer les n+1 premiers termes de la suite de Fibonacci

La **suite de Fibonacci** est une suite d'entiers très célèbre, qui apparaît dans de nombreux domaines : mathématiques, informatique, nature (Disposition des feuilles autour d'une tige (phyllotaxie), Spirales des tournesols, ananas, coquillages, Croissance des lapins (exemple donné par Fibonacci à l'origine)), art…

La suite de Fibonacci possède de nombreuses propriétés, et est notamment liée au nombre d'or. Elle apparaît dans la nature sous de nombreuses formes biologiques (ramification des arbres, disposition des feuilles sur une tige, etc.)
En biologie, elle permet aussi de modéliser l'évolution d'une population d'abeilles sur une certaine période, etc.
Cette suite est définie par la relation :
$F_{n+2} = F_{n+1} + F_n$ avec les termes initiaux : $F_0 = 0$; $F_1 = 1$; ...

On prend donc comme liste de coefficients :
a = [1, 1]
Et comme liste de premiers termes :
u = [0, 1]
Déterminons maintenant à l'aide de la fonction précédente les **6** premiers termes de la suite de Fibonacci :

```
a=[1,1] # liste des coefficients de la suite récurrente linéaire
u=[0,1] # liste des 2 premiers termes de la suite de Fibonacci
```

```
# génère les 6 premiers termes de la suite de Fibonacci
suite_fibo5 = suite_recurrente_lineaire(a,u,5)

print("6 premiers termes de la suite de Fibonacci :")
print(suite_fibo5)
```

Le code affiche :

6 premiers termes de la suite de Fibonacci :

[0, 1, 1, 2, 3, 5]

III-E Suite récurrence généralisée

III-E-1. Fonction Python

En partant de la relation de récurrence vue précédemment :

$u_{n+p} = f(u_n, u_{n+1}, ..., u_{n+p-1})$ où n et p sont des entiers positifs, et f une fonction d'une ou plusieurs variables.

On peut alors écrire la fonction Python :

```
def suite_recurrente(u,f,n):
    # génère les n+1 premiers termes de la suite définie par la relation de
récurrence Un+p = f(Un, Un+1, ..., Un+p-1)

    # nombre de termes initiaux dans u : nombre de variables de la fonction f
    p=len(u)

    # parcours des indices des termes de la suite : p -> n
    for i in range(p,n+1):
        ui = f(*u[i-p:]) # calcul du terme d'indice i à l'aide de la fonction f
        u.append(ui) # ajout du terme obtenu à la liste

    # renvoie la liste des n+1 premiers termes de la suite récurrente
    return u
```

On passe donc une fonction **f** comme argument à la fonction principale.

A noter qu'en Python on peut passer les éléments d'une liste **u** comme les arguments d'une fonction **f** en ajoutant simplement une astérisque (*) devant la liste :

```
ui = f(*u)
```

III-E-2. Évaluation de la racine carrée d'un nombre

La **suite de Héron** (ou *méthode de Héron d'Alexandrie*) est une **méthode itérative** pour approximer la racine carrée d'un nombre. C'est en fait l'un des tout premiers algorithmes connus de calcul numérique (il date d'environ 2000 ans !).

Principe :

On cherche à calculer $\sqrt{a}$ avec a > 0.

On définit une suite (u_n) par : $u_0 > 0$ (choix initial arbitraire, souvent $\mathbf{u_0=a}$ ou 1) et pour tout $u_n \geq 0$:

$$u_{n+1}=\frac{1}{2}\left(u_n+\frac{a}{u_n}\right)$$

La suite de Héron est donc définir par :

$u_{n+1} = (u_n + a/u_n)/2$ converge vers la racine carrée de **a**.

Exemple :

Calcul de √2

On choisit $u_0 = 1$.
$u_1 = (1 + 2/1)/2 = 1{,}5$
$u_2 = (1{,}5 + 2/1{,}5)/2 \approx 1{,}4167$
$u_3 = (1{,}4167 + 2/1{,}4167)/2 \approx 1{,}4142$
On obtient déjà une très bonne approximation de $\sqrt{2} \approx 1{,}41421356\ldots$
Explication :

- La suite (u_n) converge vers $\sqrt{a}$ (si $u_0 > 0$).
- L'idée vient de réécrire l'équation $x^2 = a$. Si x est une approximation de $\sqrt{a}$, alors a/x en est une autre. La moyenne $(x + a/x)/2$ est donc une meilleure approximation.
- La suite de Héron est en fait un cas particulier de la méthode de Newton appliquée à l'équation $x^2 - a = 0$.

Même exemple mais en langage Python :
Déterminons maintenant à l'aide de la fonction précédente une valeur approchée de la racine carrée de **2** correspondant au **5e** terme de la suite de Héron :

```
# fonction lambda permettant de passer du terme d'indice n à celui d'indice n+1
f = lambda x: (x + 2/x)/2

u=[1] # terme initial

# génère les 6 premiers termes de la suite de Héron
suite_heron5 = suite_recurrente(u, f, 5)

print("6 premiers termes de la suite de Héron :")
print(suite_heron5)
```

Le code affiche :

6 premiers termes de la suite de Héron :
[1, 1.5, 1.4166666666666665, 1.4142156862745097, 1.4142135623746899, 1.414213562373095]

III-E-3. Générer les n+1 premiers termes de la suite de Fibonacci
Comme on l'a déjà vu, la suite de Fibonacci est définie par la relation : $F_{n+2} = F_{n+1} + F_n$
avec les termes initiaux : $F_0 = 0$; $F_1 = 1$….
On prend donc comme liste de premiers termes :
u = [0, 1]

Déterminons maintenant à l'aide de la fonction précédente les **6** premiers termes de la suite de Fibonacci :

```
# fonction permettant d'obtenir le terme d'indice n+2 dans la suite de Fibonacci en
utilisant la relation de récurrence Fn+2 = Fn+1 + Fn
def f(*u):
    return u[0] + u[1]

u=[0, 1] # 2 premiers termes de la suite de Fibonacci

# génère les 6 premiers termes de la suite de Fibonacci
suite_fibo5 = suite_recurrente(u, f, 5)

print("6 premiers termes de la suite de Fibonacci :")
print(suite_fibo5)
```

Le code affiche :
6 premiers termes de la suite de Fibonacci :
[0, 1, 1, 2, 3, 5]

III-F. Module complet

On donne enfin le code complet permettant d'effectuer les tests sur les différentes fonctions :

```
def suite_arithmetique(u0,r,n):
    # génère les n+1 premiers termes de la suite arithmétique de raison r et de
terme initial u0

    # on initialise la variable ui et la liste u avec le terme initial de la suite
    ui = u0; u = [u0]

    # print(f'u({0}) = {u0}') # affiche u(0) = ..

    # parcours des indices des termes de la suite : 1 -> n
    for i in range(1,n+1):
        ui += r # ajout de r au terme précédent pour obtenir le terme d'indice i

        u.append(ui) # ajout du terme obtenu à la liste

        # print(f'u({i}) = {ui}') # affiche u(i) = ..

    # renvoie la liste des n+1 premiers termes de la suite arithmétique
    return u

def suite_geometrique(u0,q,n):
    # génère les n+1 premiers termes de la progression géométrique de raison q et de
terme initial u0

    # on initialise la variable ui et la liste u avec le terme initial de la suite
    ui = u0; u = [u0]

    # print(f'u({0}) = {u0}') # affiche u(0) = ..

    # parcours des indices des termes de la suite : 1 -> n
    for i in range(1,n+1):
        ui *= q # multiplication du terme précédent par q pour obtenir le terme
d'indice i

        u.append(ui) # ajout du terme obtenu à la liste

        # print(f'u({i}) = {ui}') # affiche u(i) = ..

    # renvoie la liste des n+1 premiers termes de la suite géométrique
    return u

def suite_arithmetico_geometrique(u0,a,b,n):
    # génère les n+1 premiers termes de la progression arithmético-géométrique de
paramètres a et b, et de terme initial u0

    # on initialise la variable ui et la liste u avec le terme initial de la suite
    ui = u0; u = [u0]

    # parcours des indices des termes de la suite : 1 -> n
    for i in range(1,n+1):
        ui = a*ui + b # calcul du terme d'indice i à partir du terme précédent

        u.append(ui) # ajout du terme obtenu à la liste

    # renvoie la liste des n+1 premiers termes de la suite arithmético-géométrique
    return u

def suite_recurrente_lineaire(a,u,n):
    # génère les n+1 premiers termes de la suite récurrente linéaire à partir des
listes de valeurs initiales a et u

    # nombre de constantes ai ou de termes dans la relation de récurrence
    p=len(a)

    # parcours des indices des termes de la suite : p -> n
    for i in range(p,n+1):
        # calcul du terme d'indice i
```

```
        ui = 0
        for aj,uj in zip(a,u[i-p:]):
            ui += aj*uj

        u.append(ui)

    # renvoie la liste des n+1 premiers termes de la suite récurrente linéaire
    return u

def suite_recurrente(u,f,n):
    # génère les n+1 premiers termes de la suite définie par la relation de
récurrence Un+p = f(Un, Un+1, ..., Un+p-1)

    # nombre de termes initiaux dans u : nombre de variables de la fonction f
    p=len(u)

    # parcours des indices des termes de la suite : p -> n
    for i in range(p,n+1):
        ui = f(*u[i-p:]) # calcul du terme d'indice i à l'aide de la fonction f
        u.append(ui) # ajout du terme obtenu à la liste

    # renvoie la liste des n+1 premiers termes de la suite récurrente
    return u


# Progression arithmétique du capital

print("I. Évolution du capital à intérêts simples : progression arithmétique\n")

#1er terme de la suite : capital initial
c0 = 5000

# raison de la suite : augmentation annuelle du capital de 250 euros
r = 250

# liste des valeurs successives du capital sur 10 années
valeurs_capital = suite_arithmetique(c0,r,10)

print("Evolution du capital sur 10 années :")
print(valeurs_capital)

# Progression géométrique du capital

print()
print("II. Évolution du capital à intérêts composés : progression géométrique\n")

#1er terme de la suite : capital initial
c0 = 5000

# raison de la suite géométrique : q = (1+t) avec t le taux d'intéret annuel
q = 1.05

# liste des valeurs successives du capital sur 10 années
valeurs_capital = suite_geometrique(c0,q, 10)

# liste des valeurs du capital arrondies à 2 décimales
valeurs_capital2 = [float('%.2f' % ci) for ci in valeurs_capital]

print("Évolution du capital sur 10 années :")
print(valeurs_capital2)

# Progression arithmético-géométrique
print()
print("III. Évolution du capital restant dû : progression arithmético-géométrique\
n")

#1er terme de la suite : capital initial
r0 = 10000

# raison de la suite géométrique : q = (1+t) avec t taux d'intérêt mensuel
```

```
a = 1.004
b = -1000

# liste des valeurs successives du capital restant dû sur 10 mois
valeurs_capital = suite_arithmetico_geometrique(r0,a, b, 10)

# liste des valeurs du capital restant dû arrondies à 2 décimales
valeurs_capital2 = [float('%.2f' % ci) for ci in valeurs_capital]

print("Évolution du capital restant dû sur 10 mois :")
print(valeurs_capital2)

# Suite récurrente linéaire
print()
print("IV. Suite récurrente linéaire\n")

# Suite de Fibonacci
a=[1,1] # liste des coefficients de la suite récurrente linéaire
u=[0,1] # liste des 2 premiers termes de la suite de Fibonacci

# génère les 6 premiers termes de la suite de Fibonacci
suite_fibo5 = suite_recurrente_lineaire(a,u,5)

print("6 premiers termes de la suite de Fibonacci :")
print(suite_fibo5)

print()
print("V. Suites récurrentes généralisées\n")

# calcul de la racine carrée de 2 : méthode de Héron
print("V-A. Calcul de la racine carrée de 2 - Méthode de Héron\n")

# fonction lambda permettant de passer du terme d'indice n à celui d'indice n+1
f = lambda x: (x + 2/x)/2

u=[1] # terme initial

# génère les 6 premiers termes de la suite de Héron
suite_heron5 = suite_recurrente(u, f, 5)

print("6 premiers termes de la suite de Héron :")
print(suite_heron5)

# génération des n+1 termes de la suite de Fibonacci
print()
print("V-B. Génération des n+1 termes de la suite de Fibonacci\n")

# fonction permettant d'obtenir le terme d'indice n+2 dans la suite de Fibonacci en
utilisant la relation de récurrence Fn+2 = Fn+1 + Fn
def f(*u):
    return u[0] + u[1]

u=[0, 1] # 2 premiers termes de la suite de Fibonacci

# génère les 6 premiers termes de la suite de Fibonacci
suite_fibo5 = suite_recurrente(u, f, 5)

print("6 premiers termes de la suite de Fibonacci :")
print(suite_fibo5)
```

IV. **Conclusion**

Les suites récurrentes sont donc utilisées dans des domaines très variés (en mathématiques, en économie, en biologie, etc.).

Cette présentation nous aura notamment permis de mieux les reconnaître, ensuite, comme on a pu le constater, leur implémentation en Python ne pose pas vraiment de problèmes

Ce cours est issu du site: https://www.developpez.net/forums/blogs/44027-user/b10499/suites-recurrentes-python/
Merci au site www.developpez.net pour cette source