

III

Python – Les listes

Plan du chapitre

I LISTES ET PREMIÈRES COMMANDES	2
I.1 CRÉATION D'UNE LISTE	2
I.2 LES INDICES D'UNE LISTE	2
I.3 AJOUTER UN ÉLÉMENT À UNE LISTE	3
I.4 D'AUTRES COMMANDES UTILES	6
II MODES DE GÉNÉRATION DES LISTES	7
II.1 EN EXTENSION	8
II.2 EN COMPRÉHENSION	8
III PROCÉDURES CLASSIQUES	9
III.1 PARCOURIR UNE LISTE	9
III.2 CONSERVER LES TERMES D'UNE SUITE NUMÉRIQUE	9
III.3 REPRÉSENTATION GRAPHIQUE DES TERMES D'UNE SUITE	14
III.4 CONSTRUIRE UNE LISTE AVEC UN PAS FIXÉ	15
III.5 CONSTRUIRE UNE LISTE DE VALEURS POUR UNE FONCTION	16
III.6 TRACER UNE FONCTION AVEC MATPLOTLIB	18
IV EXERCICES	20

Une **liste** permet de stocker plusieurs valeurs dans une même variable.

Les listes sont particulièrement utiles en mathématiques pour :

- conserver les termes successifs d'une suite ;
- calculer plusieurs valeurs et les conserver ;
- préparer les valeurs nécessaires au tracé d'une fonction.

LISTE DES COMMANDES Python EXIGIBLES AUX CONCOURS

- Tableau unidimensionnel ou liste.
- Définitions d'une liste avec une boucle ou en compréhension.
- Fonction range.
- Commandes append, len.
- Recherche séquentielle dans une liste.
- Commandes in del count.
- Manipulations élémentaires de listes.
- Commandes + et *.

I. LISTES ET PREMIÈRES COMMANDES

Une **liste** en **Python** est une structure de données ordonnée permettant d'emmagasiner une suite d'éléments (nombres, chaînes de caractères, etc.).

- **Création d'une liste vide** : `L = []`
- **Ajouter un élément à la fin** : `L.append(valeur)`
- **Accéder à un terme** : `L[0]` donne le premier élément, `L[1]` le deuxième, et `L[-1]` le dernier.
- **Taille d'une liste** : `len(L)` renvoie le nombre d'éléments.

Une liste est donc une collection finie d'éléments pouvant être de types différents (entiers, flottants, chaînes de caractères, listes...); ils sont notés entre crochets et séparés par des virgules.

I.1 – Création d'une liste

On crée une liste en utilisant des crochets.

Par exemple :

```
Éditeur 1 L = [3, 7, 2, 9]
```

La variable L contient alors les quatre nombres

3, 7, 2, 9.

On peut également créer une liste de nombres décimaux :

```
Éditeur 1 L = [1.2, 3.5, 4.8, 7.1]
```

Enfin, on peut créer une liste vide :

```
Éditeur 1 L = []
```

💡 Idée

Une liste vide est particulièrement utile lorsque l'on souhaite la construire progressivement.

I.2 – Les indices d'une liste

Les éléments d'une liste sont numérotés à partir de 0.

Ainsi, pour

```
Éditeur 1 L = [3, 7, 2, 9]
```

on a :

Élément	3	7	2	9
Indice	0	1	2	3

On accède à un élément en utilisant son indice entre crochets :

```
Éditeur 1 L[0]
```

donne 3 tandis que

```
Éditeur 1 L[2]
```

donne 2.

L'indice -1 permet d'accéder au dernier élément d'une liste.

Par exemple :

```
Éditeur 1 L[-1]
```

donne 9.

De même, $L[-2]$ désigne l'avant-dernier élément.

Exercice 1

On considère la liste :

```
Éditeur 1 L = [4, 8, 3, 10, 6]
```

1. Quelle est la valeur de $L[0]$?
2. Quelle est la valeur de $L[3]$?
3. Quelle est la valeur de $L[4]$?

I.3 – Ajouter un élément à une liste

Pour ajouter un élément à la fin d'une liste, on utilise la méthode `append` :

```
Éditeur 1 L = [3, 7, 2]
Éditeur 2 L.append(9)
```

La liste contient alors $[3, 7, 2, 9]$.

On peut donc construire une liste progressivement.

Exemple 1 :

On considère le programme :

💡 Idée
 Cette notation est particulièrement pratique pour les suites : lorsque les termes ont été calculés les uns après les autres, $u[-1]$ désigne toujours le dernier terme calculé.

```

Éditeur
1 L = []
2 L.append(5)
3 L.append(8)
4 L.append(2)
5 L.append(11)

```

1. Quelle est la valeur de L à la fin du programme ?
2. Quelle est la valeur de L[1] ?
3. Quelle est la valeur de L[-1] ?

Les symboles + et * permettent : soit l'addition / la multiplication de nombres, soit la concaténation / la concaténation répétée de listes.

À retenir

Les commandes
L.append(x) et L=L+[x]
font donc exactement la
même chose.

```

Éditeur
1 L=[1,2,3]
2 M=[4,5,6]
3 L+M # Renvoie la liste composée des éléments de la
4     # première puis ceux de la seconde

```

Console

```
[1, 2, 3, 4, 5, 6]
```

```

Éditeur
1 L=[1,2,3]
2 M=[4,5,6]
3 M+L

```

Console

```
[4, 5, 6, 1, 2, 3]
```

```

Éditeur
1 N=list(range(0,6))
2 N+[5,6,7]

```

Console

```
[0, 1, 2, 3, 4, 5, 5, 6, 7]
```

Remarque

La commande * fonctionne de la même façon sur les chaînes de caractères (pour répéter un message par exemple).

```

Éditeur
1 L*3 # Renvoie la liste obtenue en concaténant 3 fois
    ↪ la liste L

```

Console

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

Exemple 2 – Conserver les termes d’une suite :

Considérons la suite définie par

$$u_0 = 2 \quad \text{et} \quad u_{n+1} = 2u_n + 1.$$

On souhaite conserver dans une liste les termes

$$u_0, u_1, \dots, u_{10}.$$

On commence par placer le premier terme dans une liste :

```
Éditeur 1 u = [2]
```

Pour obtenir le terme suivant, on utilise le dernier terme connu :

```
Éditeur 1 u.append(2*u[-1] + 1)
```

On peut répéter cette opération à l’aide d’une boucle :

```
Éditeur 1 u = [2]
2 for n in range(10):
3     u.append(2*u[-1] + 1)
```

ou

```
Éditeur 1 u = [2]
2 for n in range(10):
3     u=u+[(2*u[-1] + 1)]
```

La liste u contient alors les termes

$$u_0, u_1, \dots, u_{10}.$$

renvoie la valeur de la somme des éléments de L.

```
Console
>>> u
[2, 5, 11, 23, 47, 95, 191, 383, 767, 1535, 3071]
```

⚠ Attention

Ne pas oublier les `[]` !

💡 Idée

En utilisant la commande `sum(u)` on pourra alors obtenir, par exemple, la somme

$$\sum_{n=0}^{10} u_n.$$

Exercice 2 – (Faire connaissance) [CORRECTION P. 22]

Exécuter successivement les lignes suivantes dans la console et indiquer à la suite des # l’effet de la commande.

```
Éditeur 1 L=[1,17,43]
2 type(L) #
3 L==[1,43,17] #
```

```
Éditeur 1 M=[1,2,"Truc", False, L]
2 M
```

```

Éditeur
1 [] #
2 L[0]
3 L[0]=0 #
4 L #
5 L[1] #
6 L[2] #
7 L[3] #
8 L[-1] #

Éditeur
1 type(range(0,21)) #
2 N=list(range(0,21))
3 type(N)

Éditeur
1 N[2:6] #
2 N[0:11 :2] #
3 N[::2] #
4 N[::-1] #
5 list("Truc") #

```

ATTENTION

Que dire des listes M et L après exécution de ces lignes ?

```

Éditeur
1 L=list(range(0,11))
2 L
3 M=L
4 M[0]=43
5 L

```

Il faudra donc procéder autrement si l'on souhaite copier une liste, puis modifier la copie sans modifier la liste initiale.

Pour créer une copie indépendante d'une liste L, on peut écrire `M=L[:]` par exemple.

I.4 – D'autres commandes utiles

1. La commande `len(L)` renvoie la longueur (length en anglais) de la liste L, autrement dit son nombre d'éléments.

```

Éditeur
1 L=list(range(0,10))
2 len(L)

```

Console

```

10
>>> L[len(L)-1]
9

```

⚠ Attention

Les listes L et M sont toutes deux modifiées !

💡 Idée

Cela permet d'accéder au dernier élément de la liste par `L[len(L)-1]`.

2. La commande `in` permet : soit de parcourir les éléments d'une liste quand elle est placée après une instruction `for`, soit de tester l'appartenance d'un élément à une liste (elle renvoie alors `True` ou `False` selon l'appartenance ou non).

```
Éditeur
1 L=list(range(0,10))
2 8 in L
```

```
Éditeur
1 10 in L
```

3. La commande `del L[i]` supprime l'élément indexé `i` de la liste `L` et ré-indexe la liste.

```
Éditeur
1 del L[4]
2 L
```

Console

```
[0, 1, 2, 3, 5, 6, 7, 8, 9]
```

4. La commande `L.count(x)` renvoie le nombre d'éléments de `L` ayant la valeur de `x`.

```
Éditeur
1 L=[10,12,11,10,13,10]
```

Console

```
>>> L.count(10)
3
>>> L.count(15)
0
```

5. Si `L` est une liste de nombres, la commande `sum(L)` renvoie la valeur de la somme des éléments de `L`.

Console

```
>>> sum(L)
66
```

6. Si `L` est une liste de nombres, les commandes `min(L)` et `max(L)` renvoient respectivement la valeur du minimum et du maximum des éléments de `L`.

Console

```
>>> min(L)
10
>>> max(L)
13
```

II. MODES DE GÉNÉRATION DES LISTES

En mathématiques, il y a essentiellement deux façons différentes d'écrire un ensemble :

- en extension : cela revient à décrire les éléments qu'il contient : $E = \{1; 2; 3\}$
- en compréhension : cela revient à le définir à partir d'une condition : $E = \{x \in \mathbb{R} \mid 2x+5 \geq 1\}$

C'est sensiblement identique pour générer des listes en **Python**.

II.1 – En extension

Cela revient en quelque sorte à écrire chaque élément de la liste ; ou à les ajouter à l'aide d'une boucle. Deux cas de figure :

```
Éditeur 1 L=[0,1,4,9,16,25,36,49,64,81,100]
```

Remarque

On peut aussi écrire
`L=L+[k**2]`.

ou

```
Éditeur 1 L=[]      # On commence par définir L,
2          # qui est une liste vide
3
4 for k in range(0,11):
5     L.append(k**2)
```

Console

```
>>> L [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

II.2 – En compréhension

Deux syntaxes possibles selon ce que l'on souhaite faire :

```
Éditeur 1 L=[f(x) for x in TRUC]
```

où TRUC est soit une liste, soit une commande `range()`.

Exemples 3 :

Console

```
>>> L=[k**2 for k in range(0,11)]
>>> L
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> M=[k for k in L] # Permet de copier les éléments de
>>>                  # la liste L dans M, sans créer de
>>>                  # dépendance entre les listes
>>> M
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Mais on peut aussi rajouter une condition à `x` pour qu'il soit rajouté à `L`.

```
Éditeur 1 L=[f(x) for x in TRUC if x ...]
```

Exemples 4 :

Console

```
>>> [k**2 for k in range(0,11) if k!=2]
[0, 1, 9, 16, 25, 36, 49, 64, 81, 100]

>>> [k**2 for k in range(0,11) if k!=2 if k!=3]
[0, 1, 16, 25, 36, 49, 64, 81, 100]

>>> [c*2 for c in "Test"]
['TT', 'ee', 'ss', 'tt']

>>> [i+j for i in range(0,3) for j in range(0,6)]
[0, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 2, 3, 4, 5, 6, 7]

>>> [i+j for j in range(0,6) for i in range(0,3)]
[0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6, 5, 6, 7]

>>> [i+j for i in range(0,3) for j in range(0,3) if i>=j]
[0, 1, 2, 2, 3, 4]
```

III. PROCÉDURES CLASSIQUES

III.1 – Parcourir une liste

💡 Idée

Cette structure permet notamment d'effectuer un calcul sur chacun des éléments d'une liste.

On peut parcourir successivement tous les éléments d'une liste à l'aide d'une boucle **for**.

Par exemple :

```
1 u = [2, 5, 11, 23, 47]
2
3 for terme in u:
4     print(terme)
```

La variable `terme` prend successivement les valeurs des différents éléments de la liste.

Console

```
2
5
11
23
47
```

III.2 – Conserver les termes d'une suite numérique

Pour calculer et stocker les termes d'une suite récurrente ou explicite (u_n), on initialise une liste contenant le terme initial, puis on utilise une boucle **for** pour ajouter les termes successifs avec `.append()`.

Exemple 5 – Suite arithmético-géométrique :

Considérons la suite (u_n) définie par $u_0 = 2$ et $u_{n+1} = 0,5u_n + 3$.

```

1  u = 2          # Initialisation
2
3  termes = [2]  # La liste contient u_0
4
5  # Calcul des 10 termes suivants (de u_1 à u_10)
6  for k in range(10):
7      u = 0.5 * u + 3
8      termes.append(u)
9
10 print(termes)

```

Console

```
[2, 4.0, 5.0, 5.5, 5.75, 5.875, 5.9375, 5.96875, 5.984375, 5.9921875,
5.99609375]
```

Dans cet exemple, `termes[0]` correspond à u_0 , `termes[1]` à u_1 , et plus généralement `termes[n]` à u_n .

Exercice 3 – (Conserver les termes d’une suite) [CORRECTION P. 22]

On considère la suite définie par

$$u_0 = 3 \quad \text{et} \quad u_{n+1} = u_n + 4.$$

1. Compléter le programme suivant afin de conserver les 11 premiers termes de la suite dans une liste `u`.

```

1  u = [3]
2  for n in range(10):
3      ...

```

2. Quelle est la valeur de `u[5]` ?
3. Quelle est la valeur de `u[-1]` à la fin du programme ?

Exemple 6 – Conserver les termes d’une suite :

Pour une suite définie par

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

on peut conserver les termes successifs avec :

```

1  u = [u0]
2
3  for n in range(N):
4      u.append(f(u[-1]))

```

Méthode

Les termes de la suite s'appellent par la commande `u[n]`.

Idée

`u[-1]` est le dernier terme calculé soit u_n .

Exercice 4 – (Conserver les termes d’une suite) [CORRECTION P. 23]

On considère la fonction f définie par $f(x) = \frac{1}{2} \left(x + \frac{2}{x} \right)$ et la suite

$$u_0 = 1 \quad \text{et} \quad u_{n+1} = f(u_n).$$

1. Construire une fonction f prenant en argument un réel x et renvoyant la valeur de $f(x)$.
2. Compléter le programme suivant afin de conserver les 11 premiers termes de la suite dans une liste u .

Idée

On se servira bien évidemment de la fonction f .

```
Éditeur
1 u = [...]
2 for n in range(10):
3     ...
```

3. En faisant varier le nombre de termes, quelle conjecture faites-vous sur l'éventuelle convergence de cette suite ?

1.

```
Éditeur
1 def f(x):
2     return .5*(x+2/x)
```

```
Éditeur
2 u = [1]
2
3 for n in range(10):
4     u.append(f(u[-1]))
5
6 u[-1]
```

3.

```
Console
>>> def f(x):
...     return .5*(x+2/x)
...
>>> u = [1]
>>> for n in range(10):
...     u.append(f(u[-1]))
...
>>> u[-1]
1.414213562373095

>>> u = [1]
>>> for n in range(100):
...     u.append(f(u[-1]))
...
>>> u[-1]
1.414213562373095
```

La suite $(u_n)_{n \in \mathbb{N}}$ semble converger (rapidement) vers $\sqrt{2}$.

Exercice 5 – (Calcul et somme des termes d’une suite) [CORRECTION P. 23]

On considère la suite (u_n) définie par

$$u_0 = 1 \quad \text{et} \quad u_{n+1} = 2u_n + 1.$$

On souhaite calculer les 10 premiers termes de cette suite et les stocker dans une liste U.

1. Créer une liste U contenant initialement le terme u_0 .
2. À l’aide d’une boucle **for**, calculer successivement les termes $u_1, u_2, \dots, u_9$ et les ajouter à la liste U avec **append**.
3. Afficher la liste U.
4. Utiliser la fonction **sum**(U) pour calculer la somme des 10 premiers termes de la suite.
5. Afficher cette somme.

Exercice 6 – (La suite de Syracuse – Une suite qui joue au yoyo) [CORRECTION P. 24]

On choisit le nombre $u_0 = 13$ et on définit la suite (u_n) par

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{si } u_n \text{ est impair.} \end{cases}$$

On souhaite calculer les termes de cette suite jusqu’à obtenir le nombre 1.

1. Créer une liste U contenant initialement le nombre 13.
2. À l’aide d’une boucle **while**, calculer les termes successifs de la suite et les ajouter à la liste U avec **append**.
3. Afficher la liste U.
4. Utiliser **sum**(U) pour calculer la somme de tous les termes obtenus.

À retenir

Pour tester la parité de $U[i]$ on utilisera la commande

```
if U[-1] % 2 == 0:
```

Exercice 7 – (Calcul des m premiers éléments d’une suite) [CORRECTION P. 24]

Considérons la suite (u_n) définie par $u_0 = 3$ et $u_{n+1} = \frac{2u_n}{u_n + 1}$.

1. Dans l’éditeur de texte, écrire un programme qui :
 - demande initialement à l’utilisateur d’entrer au clavier la valeur d’un entier m ,
 - crée une liste U contenant initialement l’unique élément u_0 ,
 - met à jour la liste U dans une structure itérative de sorte à ce que U contienne les valeurs des m premiers éléments de la suite (u_n) en fin de boucle.

```

1  m = ...
2  U = ...
3  for i in range(...):
4      U.append(...)
5  print(U)
```

2. Calculer les 5 premiers éléments de la suite à l’aide du programme précédent.

Exercice 8 – (ECRICOME - 2015) [CORRECTION P. 24]

L'épreuve ECRICOME - 2015 commençait par l'étude d'une suite récurrente de type $u_{n+1} = F(u_n)$. La fonction F est définie comme suit :

$$F(x) = \begin{cases} 0 & \text{si } x < 0, \\ 1 - e^{-x} & \text{si } x \geq 0 \end{cases}$$

On considère la suite $(u_n)_{n \geq 1}$ définie par $u_1 = 1$ et pour tout $n \in \mathbb{N}^*$ par : $u_{n+1} = F(u_n)$.

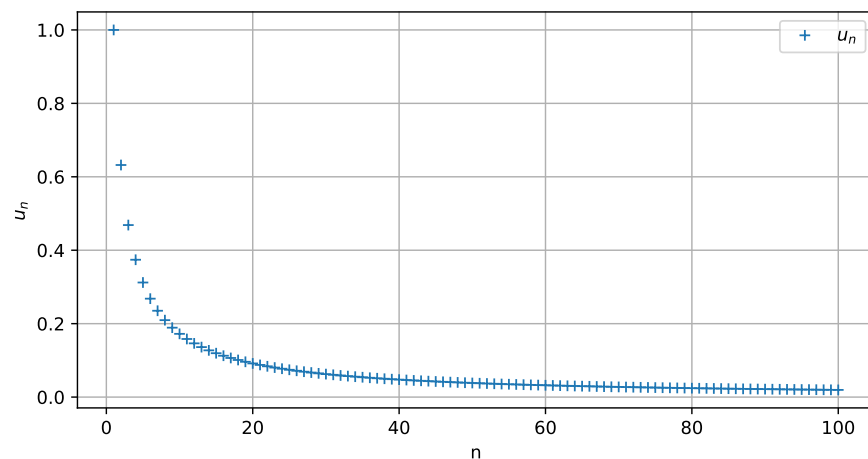
On admet que : $\forall n \geq 1, u_n > 0$.

1. Compléter le programme **Python** suivant qui permet de représenter les cent premiers termes de la suite $(u_n)_{n \geq 1}$:

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  U = np.zeros(100)
5  U[0] = 1
6
7  for n in range(99):
8      U[n+1] = ...
9
10 X = range(1, 101)
11
12 plt.plot(X, U, '+')
13 plt.show()
```

2. Le programme précédent complété permet d'obtenir la représentation graphique suivante :



Quelle conjecture pouvez-vous émettre sur la monotonie et la limite de la suite $(u_n)_{n \geq 1}$?

Exercice 9 – (Suite récurrente double) [CORRECTION P. 25]

On reprend la suite de Fibonacci du TP précédent.

Soit $(F_n)_{n \in \mathbb{N}}$ une suite définie par $F_0 = 0$, $F_1 = 1$ et, pour tout $n \in \mathbb{N}$,

$$F_{n+2} = F_{n+1} + F_n.$$

1. Définir une fonction `Fib` qui, à un entier naturel $n \geq 2$ donné en entrée, renvoie la liste contenant les n premiers termes de la suite $(F_n)_{n \in \mathbb{N}}$.

Par exemple, `Fib(7)` devra renvoyer :

```
>>>[0, 1, 1, 2, 3, 5, 8]
```

```

1  def Fib(n):
2      F = [...] # On initialise les deux
      ↪ premiers termes
3
4      for k in range(...):
5          F.append(...) # On calcule les termes
      ↪ de la suite
6
7      return F

```

2. Modifier le script précédent afin de sauver dans une liste `Phi` les valeurs de $\frac{F_{n+1}}{F_n}$ pour $n \in \mathbb{N}^*$.

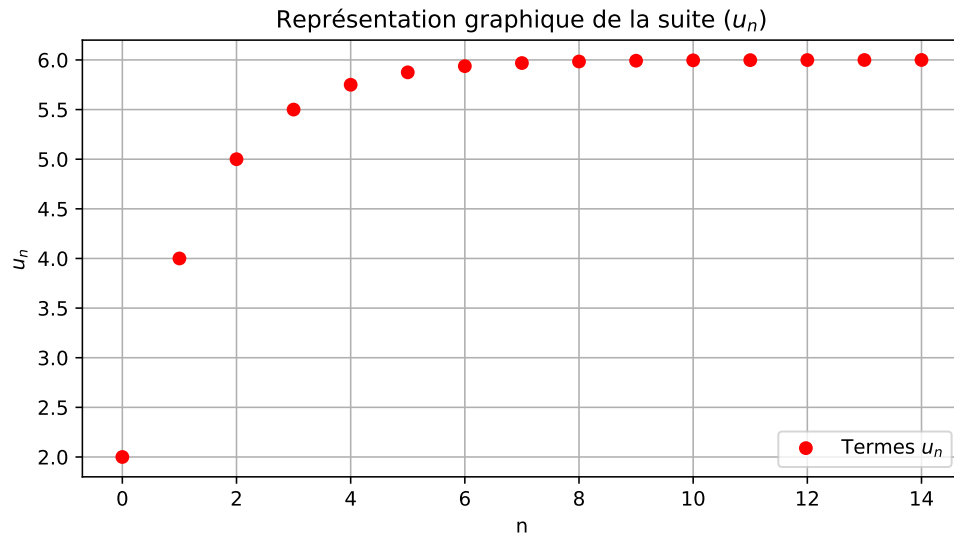
III.3 – Représentation graphique des termes d’une suite

On peut associer la liste des indices $N = [0, 1, 2, \dots]$ et la liste des termes $U = [u_0, u_1, u_2, \dots]$ pour placer les points de coordonnées (n, u_n) .

```

1  import matplotlib.pyplot as plt
2
3  u = 2
4  U = [u]
5  N = [0]
6
7  for n in range(1, 15):
8      u = 0.5 * u + 3
9      U.append(u) # On calcule les termes de la suite
10     N.append(n) # On crée la liste des indices
11
12     # Tracé des points (n, u_n)
13     plt.plot(N, U, 'ro', label="Termes u_n") # 'ro' =
      ↪ points rouges
14     plt.title("Représentation graphique de la suite u_n")
15     plt.xlabel("n")
16     plt.ylabel("u_n")
17     plt.grid(True)
18     plt.legend()
19     plt.show()

```



Exercice 10 – (En route vers le nombre d'or) [CORRECTION P. 25]

Recopier et modifier le script précédent afin qu'ils trace la représentation graphique des 15 premiers termes de la suite $\left(\frac{F_n}{F_{n-1}}\right)_{n \geq 2}$ de l'exercice (8) i.e. les points de coordonnées

$$\left(n; \frac{F_n}{F_{n-1}}\right) \text{ pour } n \geq 2.$$

```

1  import matplotlib.pyplot as plt
2
3  ...
4
5  for k in range(...):
6      ...
7
8  # Tracé des points
9  plt.plot(..., ..., 'ro', label=r"Termes Fn/F(n-1)")
10
11  plt.title(r"Représentation graphique de la suite
12  ↪ Fn/F(n-1) pour n>=2")
13  plt.xlabel("n")
14  plt.ylabel(r"Fn/F(n-1)")
15  plt.grid(True)
16  plt.legend(loc="lower right")
17  plt.show()

```

III.4 – Construire une liste avec un pas fixé

Pour obtenir des valeurs régulièrement espacées, on dit avec un pas p , on peut utiliser une boucle.

Par exemple, pour construire la liste

$$0, 0,1, 0,2, \dots, 1,$$

Idée

Cette technique est particulièrement utile pour préparer le tracé d'une fonction.

avec un pas de $\frac{1}{10}$, on peut écrire :

```
1 x = []
2
3 for n in range(11):
4     x.append(n/10)
```

On obtient alors une liste contenant 11 valeurs.

Console

```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
```

Exercice 11 – (Liste de réels à pas fixé) [CORRECTION P. 26]

On souhaite construire une liste contenant les nombres

$-2, -1,5, -1, -0,5, 0, 0,5, \dots, 2.$

1. Combien cette liste doit-elle contenir de valeurs ?
2. Compléter le programme :

```
1 x = []
2
3 for n in range(...):
4     x.append(-2 + ...)
```

3. Vérifier que $x[0]$ vaut -2 et que $x[-1]$ vaut 2 .

III.5 – Construire une liste de valeurs pour une fonction

Les listes permettent également de conserver les valeurs nécessaires à l'étude d'une fonction. On peut construire deux listes, l'une contenant les abscisses et l'autre les ordonnées :

```
1 x = []
2 y = []
3
4 for n in range(N):
5     x.append(...)
6     y.append(...)
```

Les listes permettent ainsi de **conserver les valeurs calculées** et de les réutiliser ensuite, notamment pour étudier une suite ou tracer une fonction.

Exemple 7 – Tableau de valeurs d'une fonction :

Considérons

$$f(x) = x^2 - 2x + 1.$$

On souhaite calculer les valeurs de cette fonction pour x compris entre -2 et 2 , avec un pas de $0,1$.

On commence par construire la liste des abscisses :

```

Éditeur
1 x = []
2
3 for n in range(41):
4     x.append(-2 + n/10)

Console
>>> x
[-2.0, -1.9, -1.8, -1.7, -1.6, -1.5, -1.4, -1.3, -1.2, -1.1, -1.0,
-0.8999999999999999, -0.8, -0.7, -0.6000000000000001, -
0.5, -0.3999999999999999, -0.30000000000000004, -0.1
99999999999999996, -0.10000000000000009, 0.0, 0.10000000000000009,
0.20000000000000018, 0.29999999999999998, 0.3999999999999999,
0.5, 0.6000000000000001, 0.7000000000000002, 0.7999999999999998,
0.8999999999999999, 1.0, 1.1, 1.2000000000000002,
1.2999999999999998, 1.4, 1.5, 1.6, 1.7000000000000002,
1.7999999999999998, 1.9, 2.0]

```

On construit ensuite la liste des ordonnées :

```

Éditeur
1 y = []
2
3 for valeur in x:
4     y.append(valeur**2 - 2*valeur + 1)

Console
>>> y
[9.0, 8.41, 7.84, 7.289999999999999, 6.760000000000001,
6.25, 5.76, 5.29, 4.84, 4.41, 4.0, 3.6099999999999994, 3.24,
2.8899999999999997, 2.5600000000000005, 2.25, 1.95999999
9999997, 1.6900000000000002, 1.44, 1.2100000000000002, 1.0,
0.8099999999999998, 0.6399999999999997, 0.4900000000000002,
0.3600000000000001, 0.25, 0.15999999999999992, 0.08999
999999999986, 0.040000000000000036, 0.01000000000000009,
0.0, 0.01000000000000009, 0.040000000000000036,
0.08999999999999986, 0.15999999999999992, 0.25,
0.36000000000000003, 0.4900000000000002, 0.6399999999999997,
0.81, 1.0]

```

Les deux listes contiennent alors les coordonnées des points

$$(x_0, y_0), (x_1, y_1), \dots, (x_{40}, y_{40}).$$

Idée

Elles pourront ensuite être utilisées pour tracer la courbe de la fonction.

Exercice 12 – (Préparer le tracé d'une fonction) [CORRECTION P. 26]

On considère la fonction

$$f(x) = x^3 - 2x.$$

On souhaite préparer le tracé de sa courbe sur l'intervalle $[-2, 2]$, avec un pas de 0,1.

1. Compléter le programme permettant de construire la liste des abscisses :

```

Éditeur
1  x = []
2
3  for n in range(41):
4      x.append(...)

```

2. Compléter le programme permettant de construire la liste des ordonnées :

```

Éditeur
1  y = []
2
3  for valeur in x:
4      y.append(...)

```

3. Que représentent les couples

$$(x[0], y[0]), \quad (x[1], y[1]), \quad \dots, \quad (x[-1], y[-1])?$$

Exercice 13 – (Préparer le tracé d’une fonction) [CORRECTION P. 26]

On considère la fonction

$$f(x) = x^2 - 3x + 2.$$

On souhaite calculer ses images pour les valeurs

$$x = 0, 1, 2, 3, 4.$$

1. Créer la liste `x` contenant les cinq abscisses.
2. Créer une liste vide `y`.
3. Compléter le programme afin que `y` contienne les images des nombres de la liste `x`.

```

Éditeur
1  x = [...]
2  y = ...
3
4  for valeur in x:
5      ...

```

4. Quelle est la valeur de `y[3]` ?

III.6 – Tracer une fonction avec Matplotlib

Pour représenter graphiquement une fonction f , le principe consiste à :

1. Définir la fonction en **Python**.
2. Générer une liste d’abscisses `X`.
3. Construire la liste des ordonnées correspondantes $Y = [f(x) \text{ pour } x \in X]$.
4. Utiliser la bibliothèque `matplotlib.pyplot`.

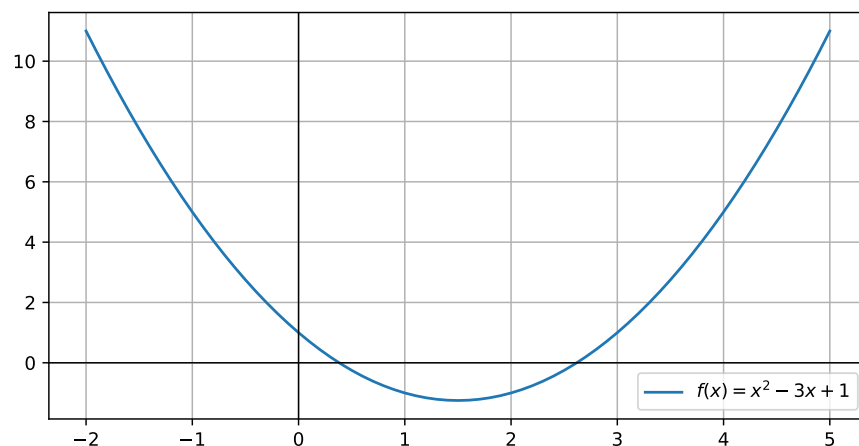
Exemple 8 – Tracé de $f(x) = x^2 - 3x + 1$ sur $[-2, 5]$:

Éditeur

```

1  import matplotlib.pyplot as plt
2
3  # 1. Définition de la fonction
4  def f(x):
5      return x**2 - 3*x + 1
6
7  # 2. Construction de la liste des abscisses (pas
   ↪ de 0.1)
8  X = []
9  x_val = -2.0
10 while x_val <= 5.0:
11     X.append(x_val)
12     x_val += 0.1
13
14 # 3. Construction de la liste des ordonnées
15 Y = []
16 for x in X:
17     Y.append(f(x))
18
19 # 4. Trace du graphique
20 plt.plot(X, Y, label="f(x) = x2 - 3x + 1")
21 plt.axhline(0, color='black', linewidth=0.8)
22 plt.axvline(0, color='black', linewidth=0.8)
23 plt.grid(True)
24 plt.legend()
25 plt.show()

```



IV. EXERCICES

Exercice 14 – $\left(\sum_{k=0}^n k^4\right)$ [CORRECTION P. 27]

1. Créer une liste L qui contient les nombres k^4 , pour $k \in \llbracket 0; n \rrbracket$, après avoir demandé à l'utilisateur de saisir une valeur pour l'entier naturel n .
2. En déduire un programme permettant de calculer $\sum_{k=0}^n k^4$, pour une valeur de n saisie par l'utilisateur.

Exercice 15 – (Suite) [CORRECTION P. 27]

Considérons la suite $(u_n)_{n \in \mathbb{N}}$ définie par :

$$\begin{cases} u_0 = 0 \\ \forall n \in \mathbb{N}, u_{n+1} = e^{-u_n}. \end{cases}$$

Créer une fonction **Python** telle que l'exécution de `listeU(n)` renvoie une liste contenant les termes u_0 à u_n de la suite $(u_n)_{n \in \mathbb{N}}$.

Exercice 16 [CORRECTION P. 27]

1. Définir une fonction `prod(L)` prenant en argument une liste L et renvoyant le produit de tous les éléments de la liste.
2. On définit la suite $(u_n)_{n \in \mathbb{N}^*}$ par

$$\forall n \in \mathbb{N}^*, \quad u_n = \prod_{k=1}^n \left(1 + \frac{1}{k}\right).$$

Construire une liste L contenant les nombres $1 + \frac{1}{k}$ pour k variant de 1 à n , l'entier n étant demandé à l'utilisateur.

3. Utiliser la fonction `prod(L)` afin de déterminer la valeur de u_{10} .

Exercice 17 – (Liste inversée) [CORRECTION P. 28]

Écrire une fonction qui prend en argument d'entrée une liste et qui renvoie la liste obtenue en inversant l'ordre des éléments.

Exercice 18 – (Suppression) [CORRECTION P. 28]

Écrire une fonction qui prend en arguments d'entrée une liste et un élément x , puis qui renvoie la nouvelle liste obtenue après suppression de toutes les occurrences de x dans cette liste.

Exercice 19 – (Occurrences) [CORRECTION P. 28]

1. Écrire une fonction prenant en arguments d'entrée une liste L et un élément x , puis renvoyant en sortie le nombre d'occurrences de x dans L.

Tester avec $L = [1, 3, 7, 3, 8, 3]$ et $x = 3$.

2. Écrire une fonction prenant en arguments d'entrée une liste L et un élément x , puis renvoyant en sortie le rang de la première occurrence de x dans L si la liste L contient l'élément x , et False sinon.

Tester avec $L = [1, 3, 7, 3, 8, 3]$ et $x=3$ puis $x=5$.

3. Écrire une fonction prenant en arguments d'entrée une liste L et un élément x , puis renvoyant en sortie le rang de la dernière occurrence de x dans L si la liste L contient l'élément x , et `False` sinon.

Tester avec $L = [1, 3, 7, 3, 8, 3]$ et $x=7$ puis $x=3$.

Exercice 20 – (Maximum) [CORRECTION P. 29]

1. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels L puis renvoyant en sortie le maximum de L .
2. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels L puis renvoyant en sortie le maximum de L ainsi que son nombre d'occurrences.
3. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels L puis renvoyant en sortie le maximum de L ainsi que le rang de sa première occurrence.

Exercice 21 – (Ordonner une liste) [CORRECTION P. 29]

Écrire une fonction qui prend en arguments d'entrée une liste de réels et qui renvoie la liste triée dans l'ordre croissant.

CORRIGÉS

Correction de l'exercice 2 : [ÉNONCÉ]

```

1  L=[1,17,43]
2  type(L)      # C'est une liste
3  L==[1,43,17] # False: l'ordre des éléments est important
4
5  M=[1,2,"Truc", False, L] # Une liste pour contenir plusieurs types
   ↳ d'objets
6
7  []           # Liste vide
8  L[0]
9  L[0]=0       # Permet de changer la valeur de l'élément indexé 0
10 L           # Donne toute la liste
11 L[1]         # 17
12 L[2]         # 43
13 L[3]         # IndexError: list index out of range
14 L[-1]        # 43: le dernier élément de L
15
16 type(range(0,21)) # range() ne renvoie pas vraiment une liste, mais
   ↳ peut être converti en liste
17 N=list(range(0,21)) # La commande list permet de convertir range() en
   ↳ liste
18 type(N)       # C'est une liste
19
20 N[2 : 6]      # Renvoie la liste composée des éléments indexés 2 à 6-1 de
   ↳ la liste N
21 N[0 : 11 : 2] # Renvoie la liste composée des éléments de N indexés de 0
   ↳ à 10, de 2 en 2
22 N[ : : 2]     # Renvoie la liste composée des éléments de N indexés de 2
   ↳ en 2
23 N[::-1]      # Renvoie la liste donc l'indexation est inversée
24 list("Truc") # ['T', 'r', 'u', 'c']

```

Correction de l'exercice 3 : [ÉNONCÉ]

```

1  u = [3]
2
3  for n in range(10):
4      u.append(u[n] + 4)
5
6  u[5]
7  u[-1]

```

Console

```

>>> u[5]
23    # c'est u5
>>> u[-1]
43    # c'est u10

```

Correction de l'exercice 4 : [ÉNONCÉ]

```

1  L=[1,17,43]
2  type(L)      # C'est une liste
3  L==[1,43,17] # False: l'ordre des éléments est important
4
5  M=[1,2,"Truc", False, L] # Une liste pour contenir plusieurs types
   ↪ d'objets
6
7  []           # Liste vide
8  L[0]
9  L[0]=0       # Permet de changer la valeur de l'élément indexé 0
10 L           # Donne toute la liste
11 L[1]         # 17
12 L[2]         # 43
13 L[3]         # IndexError: list index out of range
14 L[-1]        # 43: le dernier élément de L
15
16 type(range(0,21)) # range() ne renvoie pas vraiment une liste, mais
   ↪ peut être converti en liste
17 N=list(range(0,21)) # La commande list permet de convertir range() en
   ↪ liste
18 type(N)       # C'est une liste
19
20 N[2 :6]       # Renvoie la liste composée des éléments indexés 2 à 6-1 de
   ↪ la liste N
21 N[0 :11 :2]   # Renvoie la liste composée des éléments de N indexés de 0
   ↪ à 10, de 2 en 2
22 N[ : :2]      # Renvoie la liste composée des éléments de N indexés de 2
   ↪ en 2
23 N[: :-1]     # Renvoie la liste donc l'indexation est inversée
24 list("Truc") # ['T', 'r', 'u', 'c']

```

Correction de l'exercice 5 : [ÉNONCÉ]

```

1  U = [1]
2
3  for n in range(9):
4      U.append(2*U[-1] + 1)
5
6  print(U)
7  print(sum(U))

```

Console

```

[1, 3, 7, 15, 31, 63, 127, 255, 511, 1023]
2036

```

Correction de l'exercice 6 : [ÉNONCÉ]

```

1  U = [13]
2
3  while U[-1] != 1:
4      if U[-1] % 2 == 0:
5          U.append(U[-1] // 2)  # // 2 permet de diviser
                                ↪ des entiers
6      else:
7          U.append(3*U[-1] + 1)
8
9  print(U)
10 print(sum(U))

```

Console

```

[13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
119

```

Correction de l'exercice 7 : [ÉNONCÉ]

1.

```

1  m = int(input("Entrer un entier m : "))
2  U = [3]
3  for i in range(m):
4      U.append(2 * U[i] / (U[i] + 1))
5  print(U)

```

2. On obtient :

Console

```

[3, 1.5, 1.2, 1.0909090909090908, 1.0434782608695652,
1.0212765957446808]

```

Correction de l'exercice 8 : [ÉNONCÉ]

1.

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  U = np.zeros(100)
5  U[0] = 1
6
7  for n in range(99):
8      U[n+1] = 1 - np.exp(-U[n])
9
10 X = range(1, 101)
11
12 plt.plot(X, U, '+')
13 plt.show()

```

2. D'après la représentation graphique, la suite (u_n) semble être décroissante, de limite nulle.

Correction de l'exercice 9 : [ÉNONCÉ]

1.

```

1  def Fib(n):
2      F = [0, 1]
3
4      for k in range(2, n):
5          F.append(F[k-1] + F[k-2])
6
7      return F

```

2.

```

1  def Fib(n):
2      F = [0, 1]
3      Phi=[]
4
5      for k in range(2, n):
6          F.append(F[k-1] + F[k-2])
7          Phi.append(F[k]/F[k-1])
8
9      return Phi

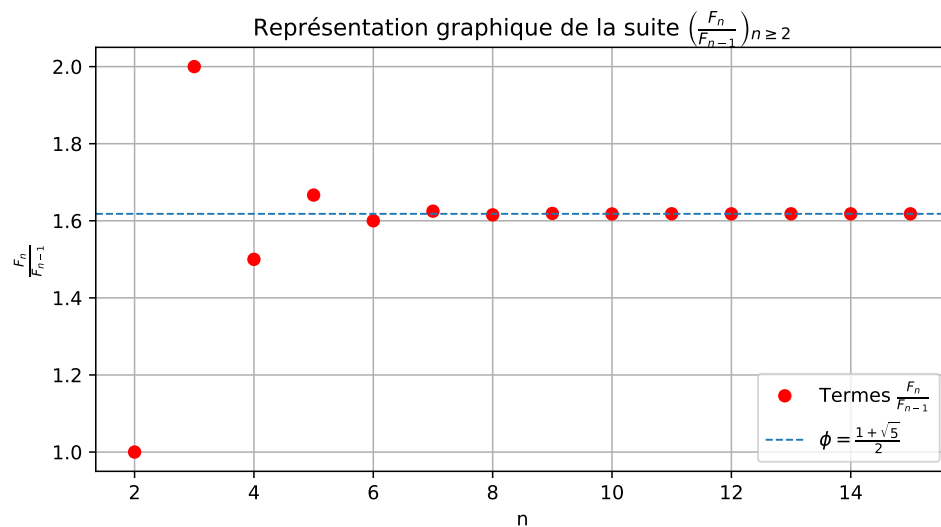
```

Correction de l'exercice 10 : [ÉNONCÉ]

```

1  import matplotlib.pyplot as plt
2
3  N=[]
4  F = [0, 1]
5  Phi = []
6
7  for k in range(2, 16):
8      F.append(F[k-1] + F[k-2]) # Calcul des Fn
9      Phi.append(F[k]/F[k-1])   # Calcul des Fn/F(n-1)
10
11     N.append(k)                # Calcul des indices
12     ↪ associées
13
14     # Tracé des points
15     plt.plot(N, Phi, 'ro', label=r"Termes Fn/F(n-1)")
16
17     plt.title(r"Représentation graphique de la suite
18     ↪ Fn/F(n-1) pour n>=2")
19     plt.xlabel("n")
20     plt.ylabel(r"Fn/F(n-1)")
21     plt.grid(True)
22     plt.legend(loc="lower right")
23
24     plt.show()

```



Correction de l'exercice 11 : [ÉNONCÉ]

```

1  x = []
2
3  for n in range(9):
4      x.append(-2 + n/2)
5
6  x[0]
7  >>>-2
8
9  x[-1]
10 >>>2

```

Correction de l'exercice 12 : [ÉNONCÉ]

1.

```

1  x = []
2
3  for n in range(41):
4      x.append(-2+n*0.1)

```

2.

```

1  y = []
2
3  for valeur in x:
4      y.append(valeur**3-2*valeur)

```

3. Les couples $(x[0], y[0]), (x[1], y[1]), \dots, (x[-1], y[-1])$ représentent les points de la représentation graphique de f pour x variant entre -2 et 2 avec un pas de 1 .

Correction de l'exercice 13 : [ÉNONCÉ]

⚠ Attention

Avant toute utilisation de la commande `append`, on pensera à initialiser la liste par un `y=[]`.

```

Éditeur
1 x = [0,1,2,3,4]
2 y = []
3
4 for valeur in x:
5     y.append(valeur**2-3*valeur+2)

```

Console

```

>>> y[3]
2

```

C'est l'image de 3 par f :

$$f(3) = 3^2 - 3 \times 3 + 2 = 2.$$

Correction de l'exercice 14 : [ÉNONCÉ]

```

Éditeur
1 n=int(input("Entrer une valeur d'un entier naturel n : "))
2 L=[k**4 for k in range(0,n+1)]
3 S=sum(L)
4 print(L)
5 print(S)

```

Correction de l'exercice 15 : [ÉNONCÉ]

```

Éditeur
1 import numpy as np
2 def listeU(n):
3     u=0
4     L=[u]
5     for k in range(1,n+1):
6         u=np.exp(-u)
7         L.append(u) # ou L=L+[u]
8     return L

```

Correction de l'exercice 16 : [ÉNONCÉ]

1. On parcourt les éléments de la liste et on les multiplie successivement.

```

Éditeur
1 def prod(L):
2     p = 1
3     for x in L:
4         p = p*x
5     return p

```

2. On demande la valeur de n , puis on construit la liste des facteurs

$$1 + \frac{1}{k}, \quad k = 1, \dots, n.$$

```

Éditeur
1 n = int(input("n = "))
2
3 L = []
4 for k in range(1,n+1):
5     L.append(1+1/k)
6
7 print(L)

```

3. Pour $n = 10$, on utilise la fonction `prod(L)`.

```

1  n = 10
2
3  u = prod(L)
4  print(u)

```

On obtient

$$u_{10} = 11.$$

Correction de l'exercice 17 : [ÉNONCÉ]

```

1  def listeinversee(L):
2      M=[]
3      for k in L:
4          M=[k]+M # permet d'ajouter l'élément par la gauche
5      return M

```

Correction de l'exercice 18 : [ÉNONCÉ]

```

1  def suppression(L,x):
2      M=[k for k in L if k!=x] # Nouvelle liste avec les
   ↪ éléments différents de x
3      return M

```

Correction de l'exercice 19 : [ÉNONCÉ]

1.

```

1  def nombreocc(L,x):
2      c=0
3      for e in L:
4          if e==x:
5              c=c+1
6      return c

```

2.

```

1  def premiereocc(L,x):
2      for k in range(0,len(L)):
3          if L[k]==x:
4              return k
5      return False

```

3.

```

1  def derniereocc(L,x):
2      rang=False
3      for k in range(0,len(L)):
4          if L[k]==x:
5              rang=k
6      return rang
7  def derniereoccbis(L,x):
8      return len(L)-1-premiereocc(L[::-1],x)

```

Correction de l'exercice 20 : [ÉNONCÉ]

1.

```

1  def monmax(L):
2      maxi=L[0]
3      for x in L:
4          if x>maxi:
5              maxi=x
6      return maxi
7
8  def monmaxbis(L): #ou en parcourant les rangs
9      maxi=L[0]
10     for k in range(0,len(L)):
11         if L[k]>maxi:
12             maxi=L[k]
13     return maxi

```

2.

```

1  def monmax(L):
2      maxi=L[0]
3      c=0
4      for x in L:
5          if x>maxi:
6              maxi=x
7              c=1
8          elif x==maxi:
9              c=c+1
10     return maxi,c

```

3.

```

1  def monmax(L):
2      maxi=L[0]
3      rang=0
4      for k in range(1,len(L)):
5          if L[k]>maxi:
6              maxi=L[k]
7              rang=k
8     return maxi,rang

```

Correction de l'exercice 21 : [ÉNONCÉ]

```

1  def listeordonnee(L):
2      for k in range(1,len(L)):
3          for j in range(0,k):
4              if L[j]>L[k]:
5                  L[j],L[k]=L[k],L[j]
6     return L

```