

Séance 7 : les listes en Python : instructions de bases

1°/ Constituer une liste

Définition : Une liste est une ensemble ordonné d'objets de types divers.

En Python : Pour ranger (ou affecter) trois objets (on parlera de termes) 5, un et 6,3 dans une liste notée L, on utilise l'instruction : `L = [ 5 , "un" , 6.3 ]`

Dans cet exemple, la liste L est définie en indiquant chacun de ses éléments.

On dit que la liste L *est définie en extension*.

Le premier terme de cette liste est 5, de type **int** (entier), de deuxième est "un", de type **str** (chaîne de caractères) et le troisième est 6,3 de type **float** (flottant).

Dans une liste chaque terme occupe un certain *rang* (on parlera aussi d'*indice*).

Attention : les rangs sont numérotés à partir de 0.

Le rang du premier terme est donc 0 ! Dans notre exemple, le terme 6,3 a pour rang 2.

La **longueur d'une liste L** est son nombre d'éléments. Elle est obtenue par la fonction : `len(L)`.

Pour créer la liste L des entiers compris entre a et b (a et b entiers) : `L = list(range( a , b + 1))`.

Liste définie en compréhension : on définit une liste par une propriété caractéristique ou un test
Soit L une liste donnée. Alors :

- La liste `[f(x) for x in L]` est la liste créée à partir de L et formée des éléments f(x), pour x parcourant la liste L.
- La liste `[x for x in L if test T]` est la liste formée des éléments de L qui vérifient le test T.

Exemples : Soit la liste `L = [ 1, 2, 3, 4, 5, 6 , 7, 8, 9 ]`.

- La liste des carrés des entiers de 1 à 9 est définie par `E = [i**2 for i in L]` ou encore `E = [i**2 for i in range(1,10)]` (sans référence cette fois à la suite L).
- La liste S des entiers de la liste L supérieurs à 3 est définie par `S = [x for x in L if x > 3]`

Remarques : • `L = []` crée une liste vide.

- pour créer une liste contenant, par exemple, cinquante « 0 » : `L = [0]*50`

2°/ Afficher une liste

Pour afficher la liste L en entier, on utilise `print(L)`.

Pour afficher le terme "un" de la liste précédente, il faut afficher le terme de rang 1.

On utilise `print(L[1])`.

Pour afficher les termes de rang 1 à 2, on utilise `L [1 : 3]`.

Dans notre exemple, on obtient alors ["un", 6.3].

3°/ Modifier une liste

Dans une liste, on va pouvoir :
modifier, ajouter, supprimer un terme et inverser l'ordre des termes.

a) Modifier un terme

Pour que le terme de rang 2 prenne la valeur 24, on tape : `L[2] = 24`

On a alors, dans notre exemple : `print(L) = [ 5 , "un", 24]`

b) Ajouter des termes

Pour introduire le terme 7 à la fin de la liste, on tape : `L = L + [7]` ou `L.append(7)`

On a alors, dans notre exemple : `print(L) = [ 5 , "un", 24 , 7 ]`.

Pour introduire les termes 6 et 5 à la fin de la liste, on tape : `L = L + [6 , 5]`

On a alors, dans notre exemple : `print(L) = [ 5 , "un", 24 , 7 , 6 , 5 ]`.

On dit alors que *la liste est définie par ajouts successifs*.

c) Supprimer des termes

Pour supprimer le terme de rang 2 de la liste L, on utilise : `del L[2]` ou `L.pop(2)`

Dans le second cas, le terme supprimé est alors affiché.

On a alors, dans notre exemple : `print(L) = [ 5 , "un", 7 , 6 , 5 ]`.

Pour enlever de la liste le premier terme égale à i : `L.remove(i)`

Remarques : `del L[1 : 3]` supprimera les termes de rang 1 et 2.

`del L[ : ]` videra la liste de tous ces éléments.

Astuce : • pour accéder au dernier terme de la liste, taper `L[-1]`.

• pour accéder à l'avant dernier terme de la liste, taper `L[-2]`. Etc.

d) Insérer l'ordre des termes

Pour inverser par exemple l'ordre des termes de rang 2 et de rang 3 de la liste L, on utilise :

`L[2] , L[3] = L[3] , L[2]`

Pour trier une liste L par ordre croissant (ou par ordre alphabétique) on utilise : `L1 = sorted(L)`.

Pour trier une liste L par ordre décroissant on utilise : `L2 = list(reversed(L1))`.

Remarque : on a créé chaque fois une nouvelle liste : L1 puis L2.

e) Insertion de termes et informations sur les éléments d'une liste

• Pour insérer au rang i les termes de la liste [2 , 3] dans la liste L, on tape : `L[i : i] = [2 , 3]`.

• Pour insérer au rang i le terme « oui » dans la liste L, on tape : `L.insert(i , « oui »)`.

• Pour afficher le plus grand nombre contenu dans une liste L, on tape : `print(max(L))`.

• Pour afficher le plus petit nombre contenu dans une liste L, on tape : `print(min(L))`.

→ 😊 Coup de pouce vidéo : https://www.youtube.com/watch?v=AR_p-yO-wGI
et livre page 16

4°/ Tester si un élément appartient à une liste

Pour savoir, par exemple, si le nombre 6 appartient à la liste L, on tape : `print( 5 in L)`.

La valeur affichée sera True si 6 appartient à la liste et False sinon.

Pour savoir si le nombre "un" appartient à la liste L, on tape : `print("un" in L)`.

La valeur affichée sera True si "un" appartient à la liste et False sinon.

Dans notre exemple, on obtiendra donc True dans les deux cas.

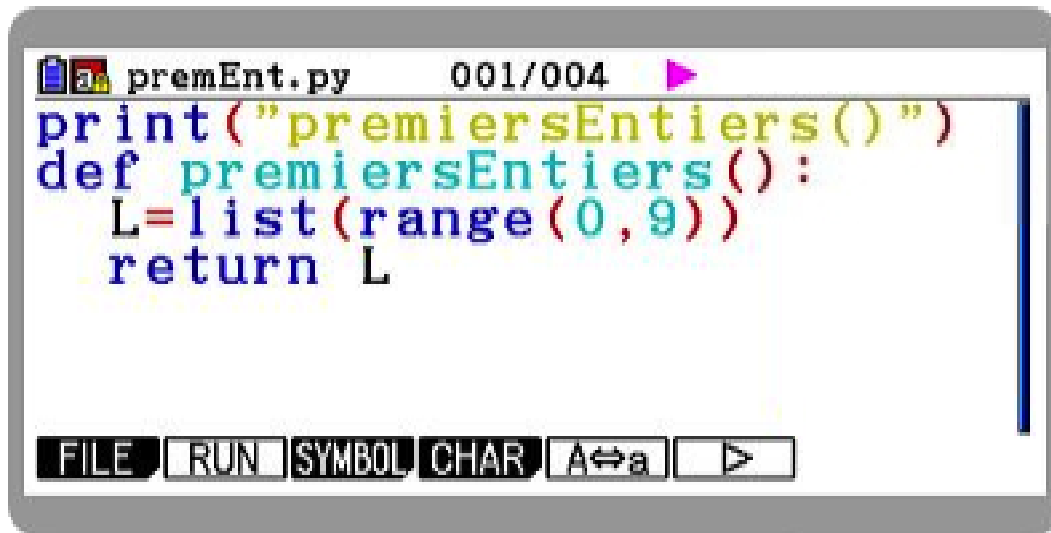
5°/ Opérations sur les listes

On peut ajouter aux termes d'une première liste **L1** ceux d'une deuxième liste **L2** en tapant : **L1+L2**

On peut reproduire trois fois, successivement, les termes d'une liste **L** en tapant : **L*3**

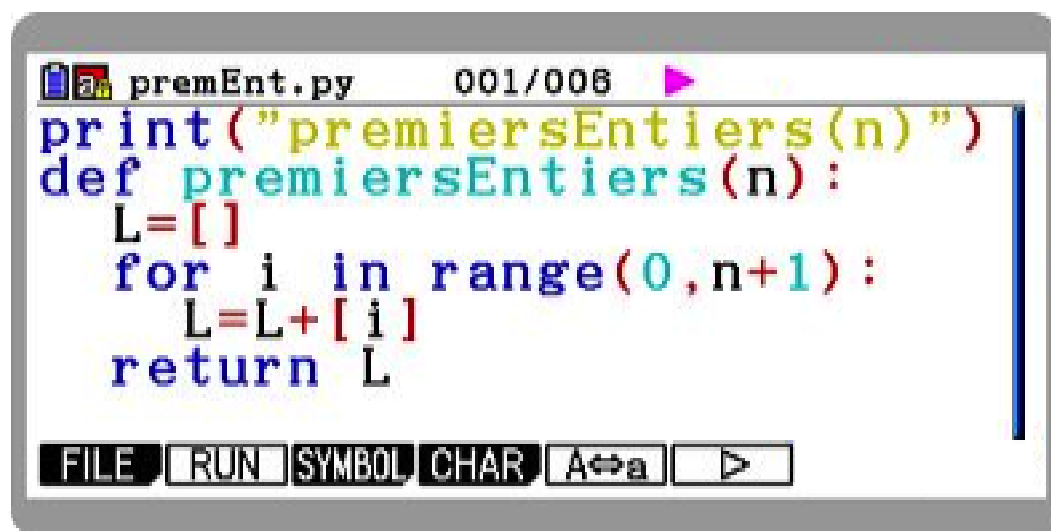
→ ☺ Coup de pouce vidéo : <https://www.youtube.com/watch?v=MAyxQmQd1CE>

Exemple : On souhaite créer une liste contenant les huit premiers entiers naturels.
Écrire un script, *premEnt*, sur ta calculatrice, correspondant à ce problème et en utilisant l'instruction **list(range(a , b + 1))**.



```
premEnt.py    001/004
print("premiersEntiers()")
def premiersEntiers():
    L=list(range(0,9))
    return L
```

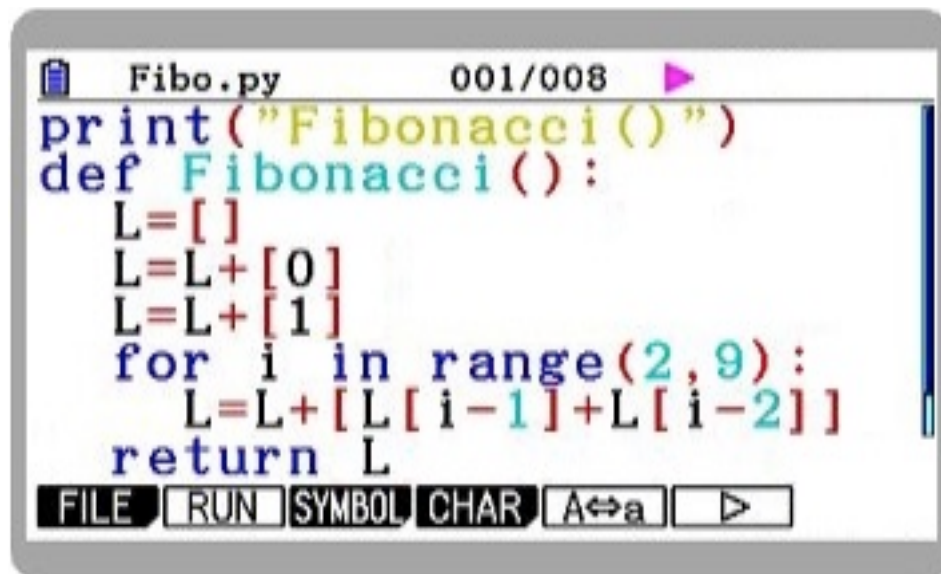
Modifier le script afin de pouvoir créer la liste des *n* premiers entiers naturels et sans utiliser l'instruction **list(range(a , b + 1))**.



```
premEnt.py    001/008
print("premiersEntiers(n)")
def premiersEntiers(n):
    L=[]
    for i in range(0,n+1):
        L=L+[i]
    return L
```

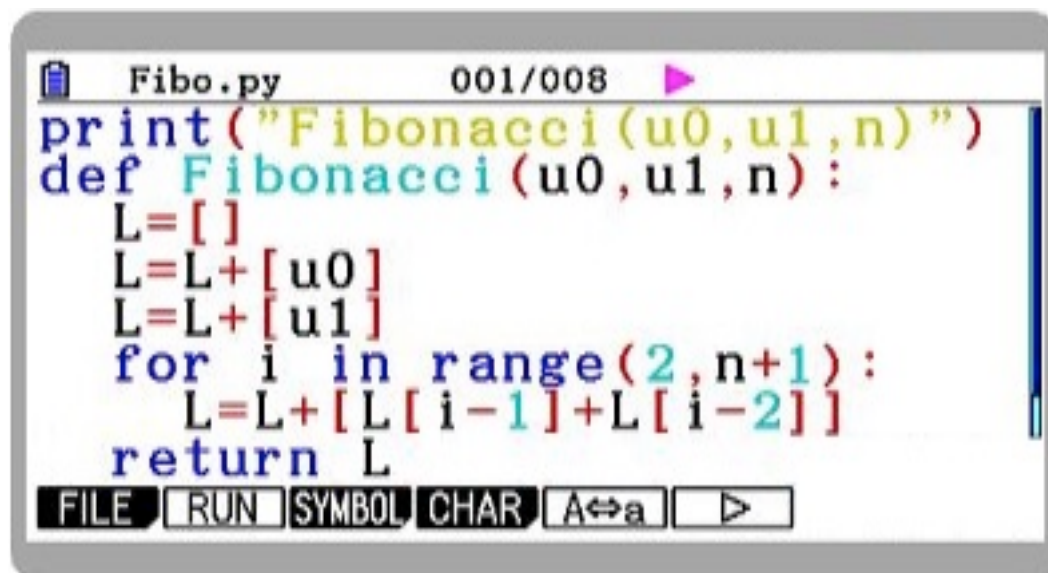
Application : Écrire un script, *Fibo*, sur ta calculatrice, inscrivant dans une liste les 10 premiers termes de la suite de Fibonacci de premiers termes 0 et 1, puis affichant la liste.

Modifier le script afin de pouvoir faire varier les deux premiers termes et le nombre de termes affichés.



The screenshot shows a Python IDE window titled 'Fibo.py' with a line counter '001/008'. The code defines a function 'Fibonacci()' that initializes a list 'L' with [0, 1] and then iterates from 2 to 9, calculating the next Fibonacci number as the sum of the two previous numbers. The script prints the result of 'Fibonacci()'. The IDE interface includes a menu bar with 'FILE', 'RUN', 'SYMBOL', and 'CHAR', and a status bar with 'A↔a' and a play button.

```
print("Fibonacci()")
def Fibonacci():
    L=[]
    L=L+[0]
    L=L+[1]
    for i in range(2,9):
        L=L+[L[i-1]+L[i-2]]
    return L
```



The screenshot shows a Python IDE window titled 'Fibo.py' with a line counter '001/008'. The code defines a function 'Fibonacci(u0, u1, n)' that initializes a list 'L' with [u0, u1] and then iterates from 2 to n+1, calculating the next Fibonacci number as the sum of the two previous numbers. The script prints the result of 'Fibonacci(u0, u1, n)'. The IDE interface includes a menu bar with 'FILE', 'RUN', 'SYMBOL', and 'CHAR', and a status bar with 'A↔a' and a play button.

```
print("Fibonacci(u0,u1,n)")
def Fibonacci(u0,u1,n):
    L=[]
    L=L+[u0]
    L=L+[u1]
    for i in range(2,n+1):
        L=L+[L[i-1]+L[i-2]]
    return L
```