ALGO 04 – LES LISTES

Une **liste** est une succession d'éléments rangés dans un certain *ordre*, séparés par des virgules et délimités par des crochets. Une liste peut contenir des éléments de types différents, comme des entiers, des chaînes de caractères, des flottants, des booléens...

I Opérations sur les listes

On considère pour chacun des exemples la liste suivante.

Entrée [1]:
```
L = ["Lundi", 12, "Info", 2.4]
```

Syntaxe	Explication	Exemple
`L=[]`	Créer une liste vide	
`len(L)`	Renvoie la longueur d'une liste	`len(L)` renvoie `4`
`L[i]`	Renvoie l'élément d'indice i*	`L[2]` renvoie `"Info"`
`L[deb:fin]`	Renvoie la liste des éléments d'indice `deb` à `fin-1`	`L[0:2]` renvoie `["Lundi", 12]`
`L[deb:]`	Renvoie la liste des éléments d'indice compris entre `deb` et le dernier indice	`L[2:]` renvoie `["Info, 2.4]`
`L[:fin]`	Renvoie la liste des éléments d'indice compris entre le premier indice et `fin-1`	`L[:2]` renvoie `["Lundi",12]`
`x in L`	Renvoie `True` si `x` est dans `L` et `False` sinon	`12 in L` renvoie `True`
`L[i]=x`	Remplace l'élément d'indice i par `x` (pas d'affichage)	`L[2]="Maths"` modifie la liste qui devient `L = ["Lundi", 12, "Maths", 2.4]`
`L.append(x)`	Ajoute l'élément `x` à la fin de la liste (pas d'affichage)	`L.append(1)` modifie la liste qui devient `L = ["Lundi", 12, "Info", 2.4, 1]`
`L.pop()`	Supprime le dernier élément de la liste (pas d'afficage)	`L.pop()` modifie la liste qui devient `L = ["Lundi", 12, "Info"]`
`del L[i]`	Supprime l'élément en position i	`del L[2]` modifie la liste qui devient `L = ["Lundi", 12, 2.4]`
`L1+L2`	Concatène les deux listes	`[1,2]+["Maths", 3]` renvoie `[1,2, "Maths", 3]`
`L*k`	Répète k fois la liste par concaténation	`[1,2]*3` renvoie `[1,2, 1,2, 1,2]`

*Chaque élément de la liste (et plus généralement d'une séquence) est *indicé* par un entier indiquant sa position. Dans une liste de longueur n, les éléments sont **indicés à partir de 0**, le dernier élément a donc pour indice $n-1$.

II Mode de génération des listes

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

- En **extension**. Cela revient à décrire un par un les éléments qu'il contient, par exemple $E = \{1, 2, 3\}$.
- En **compréhension**. Cela revient à définir l'ensemble à partir d'une condition, par exemple $E = \{x \in \mathbb{R} \mid x \geqslant 1\}$.

C'est sensiblement pareil pour générer des listes en Python.

II.1 Définition d'une liste en extension

Cela revient à écrire chaque élément de la liste...

Entrée [2]:
```
L = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

...ou à les ajouter à l'aide d'une boucle

Entrée [3]:
```
L = []
for k in range(0,11):
    L.append(k**2)
print(L)
```

Out [3]:
```
L = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

II.2 Définition d'une liste en compréhension

La syntaxe générale pour définir une liste en compréhension est la suivante :

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

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

Entrée [4]:
```
L = [k**2 for k in range(0,11)]
print(L)
```

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

Entrée [5]:
```
M = [k**2 for k in range(0,11) if k !=2 if k!=3]
print(M)
```

Out [5]:
```
[0, 1, 16, 25, 36, 49, 64, 81, 100]
```

Entrée [6]:
```
N = [i+j for i in range(0,3) for j in range(0,6)]
print(N)
```

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

Entrée [7]:
```
P =[i+j for i in range(0,3) for j in range(0,3) if i>=j]
print(P)
```

Out [7]:
```
[0, 1, 2, 2, 3, 4]
```

III Lecture de programme

Pour chaque programme, noter dans chaque case du tableau les valeurs successives prises par les variables et s'il y a un affichage, écrire la valeur affichée ou faire une croix sinon.

(i)

```
L1 = [5, 4, 2]
L2 = [True, "Oui", 4]
print(L1[1])
print(L2[2])
print( L1 + [-1, 0])
print(L1 * 2)
print(L1[:2])
print(4 in L1)
print(-10 in L1)
```

	L1	L2	Affichage
Ligne 1	[5,4,2]		
Ligne 2	[5,4,2]	[True,"Oui",4]	
Ligne 3	[5,4,2]	[True,"Oui",4]	4
Ligne 4	[5,4,2]	[True,"Oui",4]	4
Ligne 5	[5,4,2]	[True,"Oui",4]	[5,4,2,-1,0]
Ligne 6	[5,4,2]	[True,"Oui",4]	[5,4,2,5,4,2]
Ligne 7	[5,4,2]	[True,"Oui",4]	[5,4]
Ligne 8	[5,4,2]	[True,"Oui",4]	True
Ligne 9	[5,4,2]	[True,"Oui",4]	False

(ii)

```
L = [-1, 2, 8, 4, 6]
print(L[1:3])
x = L[2]
print( x * 2)
y = x * 3
L.append(y)
print(L)
L[1:]
```

	L	x	y	Affichage
Ligne 1	[-1,2,8,4,6]			
Ligne 2	[-1,2,8,4,6]			[2,8]
Ligne 3	[-1,2,8,4,6]	8		
Ligne 4	[-1,2,8,4,6]	8		16
Ligne 5	[-1,2,8,4,6]	8	24	
Ligne 6	[-1,2,8,4,6, 24]	8	24	
Ligne 7	[-1,2,8,4,6, 24]	8	24	[-1,2,8,4,6, 24]
Ligne 8	[-1,2,8,4,6, 24]	8	24	[2,8,4,6, 24]

(iii)

```
L = []
print(L)
L.append("Bonjour")
L = L + [1,2]
print(L)
L[1] = 3
print(L)
L.pop()
```

	L	Affichage
Ligne 1	[]	
Ligne 2		[]
Ligne 3	["Bonjour"]	
Ligne 4	["Bonjour",1,2]	
Ligne 5		["Bonjour",1,2]
Ligne 6	["Bonjour",3,2]	
Ligne 7		["Bonjour",3,2]
Ligne 8	["Bonjour",3]	

(iv)

```
L = ["A", "M", "T", "H"]
L[0] = "I"
print(L)
x = 0
k = x + 2
print(L[k])
L[1] = L[3]
print(L)
del L[1]
```

	L	x	k	Affichage
Ligne 1	["A", "M", "T", "H"]			
Ligne 2	["I", "M", "T", "H"]			
Ligne 3	["I", "M", "T", "H"]			["I", "M", "T", "H"]
Ligne 4	["I", "M", "T", "H"]	0		
Ligne 5	["I", "M", "T", "H"]	0	2	
Ligne 6	["I", "M", "T", "H"]	0	2	"T"
Ligne 7	["I", "H", "T", "H"]	0	2	
Ligne 8	["I", "H", "T", "H"]	0	2	["I", "H", "T", "H"]
Ligne 9	["I", "T", "H"]	0	2	

(v)

```
L = [-1, 100, 2.0, 3, 6]
n = len(L)
print(n)
print(L[n-1])
print(L[0]!=L[2])
x = L[0]
y = L[2]
print(x > y)
```

	L	n	x	y	Affichage
Ligne 1	[-1, 100, 2, 3, 6]				
Ligne 2	[-1, 100, 2, 3, 6]	5			
Ligne 3	[-1, 100, 2, 3, 6]	5			5
Ligne 4	[-1, 100, 2, 3, 6]	5			6
Ligne 5	[-1, 100, 2, 3, 6]	5			True
Ligne 6	[-1, 100, 2, 3, 6]	5	-1		
Ligne 7	[-1, 100, 2, 3, 6]	5	-1	2	
Ligne 8	[-1, 100, 2, 3, 6]	5	-1	2	False

(vi)

```
L1 = [0, 3, 5]
L2 = [4]
L3 = L1 + L2
print(L3, len(L3))
a, b = L3[0], L3[3]
print(a + b)
L3 = L2 * 5
print(L3, len(L3))
```

	L1	L2	L3	a	b	Affichage
Ligne 1	[0,3,5]					
Ligne 2	[0,3,5]	[4]				
Ligne 3	[0,3,5]	[4]	[0,3,5,4]			
Ligne 4	[0,3,5]	[4]	[0,3,5,4]			
Ligne 5	[0,3,5]	[4]	[0,3,5,4]	0	4	[0,3,5,4] 4
Ligne 6	[0,3,5]	[4]	[0,3,5,4]	0	4	4
Ligne 7	[0,3,5]	[4]	[4,4,4,4]	0	4	
Ligne 8	[0,3,5]	[4]	[4,4,4,4]	0	4	[4,4,4,4] 5

(vii) Pour chaque couple de listes `L1` et `L2`, indiquer l'affichage du programme.

```
#On suppose que les deux listes
#L1 et L2 sont connues
if len(L1) == len(L2) :
    print("Meme longueur")
elif len(L1) < len(L2) :
    print(len(L2))
else :
    print(len(L1))
```

L1	L2	Affichage
[1,6,3]	["Oui",12]	3
["a","b"]	["c","d"]	Meme longueur
[0]	[6,7,8,9]	4

(viii) Pour chaque liste `L`, indiquer l'affichage du programme.

```
#On suppose que
#la liste L connue
n = len(L)
if n > 4 :
    print(L[n-1])
elif n == 4 :
    print(L[n-2])
else :
    print(L[0])
```

L	Affichage
[1, 6, 3, 12]	3
["a", 1, "b", 2, "c", 3]	3
[0]	0

IV Exercices

Exercice 1

1. Créer une liste L1 contenant les éléments 4, 7, 12, 11 et 8 dans cet ordre.

```
Entrée [8]: L1 = [4, 7, 12, 11, 8]
```

2. Afficher le nombre d'éléments dans la liste L1.

```
Entrée [9]: len(L1)

 Out [9]: 5
```

3. Afficher le premier élément de L1.

```
Entrée [10]: L1[0]

 Out [10]: 4
```

4. Afficher le dernier élément de L1.

```
Entrée [11]: L1[len(L1)-1]

 Out [11]: 8
```

5. Afficher le troisième élément de L1 (le 12).

```
Entrée [12]: L1[2]

 Out [12]: 12
```

6. Afficher la portion de liste [7,12] à partir de L1.

Entrée [13]:
```
L1[1:3]
```

Out [13]:
```
[7,12]
```

7. Ajouter un 13 à la fin de L1.

Entrée [14]:
```
L1.append(13)
```

8. Vérifier, avec un booléen, que 13 est dans la liste.

Entrée [15]:
```
13 in L1
```

Out [15]:
```
True
```

9. Supprimer le 13.

Entrée [16]:
```
L1.pop()
```

10. Vérifier, avec un booléen, que 13 n'est plus dans la liste.

Entrée [17]:
```
13 in L1
```

Out [17]:
```
False
```

11. Modifier le deuxième élément (le 7) pour qu'il soit égal à -1.

Entrée [18]:
```
L1[1] = -1
```

12. Afficher la liste.

Entrée [19]:
```
print(L1)
```

Out [19]:
```
[4,-1, 12,11,8]
```

Exercice 2

1. Créer la liste L2 = [3,6,9,12,...,60] en une instruction. *On pourra remarquer que tous les éléments de la liste sont de la forme $3 \times k$ avec k un entier.*

Entrée [20]:
```
L2 = [3*k for k in range(1,21)]
```

2. Définir maintenant la liste L2 en partant d'une liste vide [ ] en utilisant une boucle `for` et des `append` successifs.

Entrée [21]:
```
L2 = []
for k in range(1,21):
    L2.append(3*k)
```

3. Enfin, définir L2 en partant d'une liste contenant que des zéros puis modifier successivement les éléments à l'aide d'une boucle `for`.

Entrée [22]:
```
L2 = [0]*20
for k in range(1,21):
    L2[k] = 3*(k+1)
```

Exercice 3

1. Définir une fonction Python, qui prend en argument un entier naturel `n` et qui renvoie la liste qui contient les nombres k^4 pour tout $k \in \{0, \dots, n\}$.

Entrée [23]:
```
def listeentierpuissance4(n):
    L = [k**4 for k in range(0,n+1)
    return(L)
```

2. Définir une fonction Python, qui prend en argument un entier naturel `n` et qui renvoie la valeur de la somme suivante

$$\sum_{k=0}^{n} k^4.$$

On pourra utiliser la commande `sum` *qui permet de sommer tous les éléments d'une liste.*

Entrée [24]:
```
def sommeentierpuissance4(n):
    L = [k**4 for k in range(0,n+1)
    S = sum(L)
    return(S)
```