Les listes en Python

Site Internet : **www.gecif.net**	Type de document : **Fiche pratique**	Intercalaire :	Date :

I – Écriture d'une liste en Python

En Python une liste est une variable contenant une collection de valeurs. Une liste s'écrit entre crochets et peut contenir **des objets de différents types**. Une liste en Python désigne une collection **finie et ordonnée** d'objets. Elle est la représentation d'un ensemble fini d'objets que l'on distingue par leur **ordre d'apparition**. On crée une liste à l'aide **des crochets [ ]** et on sépare les objets de cette liste à l'aide d'une **virgule**. Exemples :

```
liste=[1,2,3,4]
```

```
liste=["Python",2000,'R',[1,2,3],89,45,'Fin !']
```

Remarque : la liste vide existe et elle se note `[]`. La liste vide est la seule liste qui ne possède aucun élément.

II – Accès aux éléments d'une liste

L'accès à un élément de la liste de fait **par indexation à partir de zéro** :

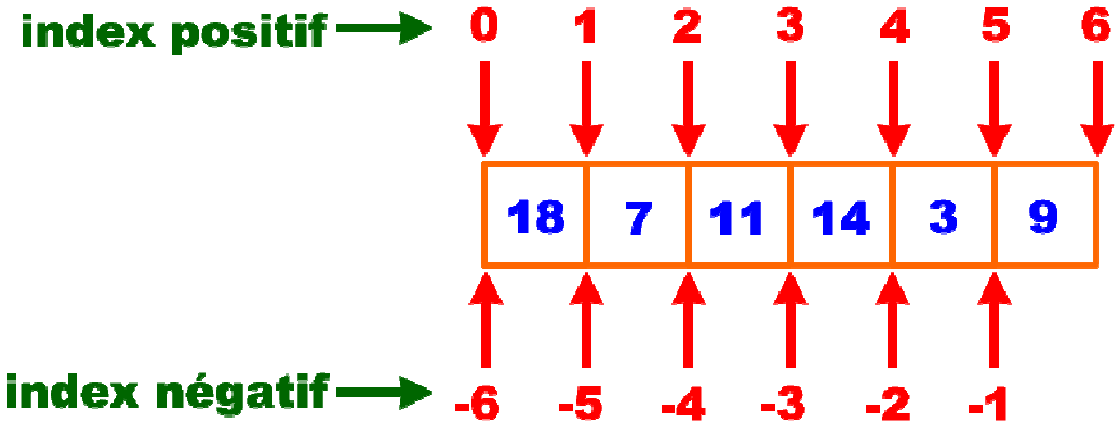
```
>>> liste=[1,2,3,4,5,6]
>>> liste[0]
1
>>> liste[1]
2
>>> liste[5]
6
>>> liste[3]
4
```

Un index négatif permet de partir de la fin de la liste, **liste[-1]** étant le **dernier** élément de la liste :

```
>>> liste=[1,2,3,4,5,6]
>>> liste[-1]
6
>>> liste[-2]
5
>>> liste[-5]
2
>>> liste[-6]
1
```

Python dispose d'une syntaxe à la fois simple et puissante appelée **tranchage** pour extraire une sous-liste d'une liste, c'est-à-dire sélectionner certains éléments de la liste **à partir d'indexs**, qui peuvent être soit positifs soit négatifs.

Le tranchage consiste à couper la liste "en tranche" et à en extraire certaines. Pour bien comprendre la technique d'extraction d'une sous-liste par tranchage, il faut imaginer la liste comme étant un saucisson, les indexs étant des coupes et les éléments de la liste étant les tranches.



Exemples en partant de la liste ci-dessus : `liste=[18,7,11,14,3,9]`

Question : quelle syntaxe permet d'extraire la sous-liste `[7, 11, 14, 3]` de la liste ci-dessus ?
Réponse : `liste[1:5]` car les éléments à extraire **sont encadrés par les indexs 1 et 5**

Question : quelle syntaxe permet d'extraire la sous-liste `[14, 3]` ?
Réponse : `liste[3:5]` car les éléments à extraire sont encadrés par les indexs 3 et 5

Question : quelle syntaxe permet d'extraire la même sous-liste `[14, 3]` mais en utilisant des indexs négatifs ?
Réponse : `liste[-3:-1]` car les éléments à extraire sont encadrés par les indexs négatifs -3 et -1

Question : quelle syntaxe donnera la sous-liste `[7, 11, 14]` en utilisant un index positif et un index négatif ?
Réponse : il y a cette fois 2 solutions : `liste[1:-2]` et `liste[-5:4]` qui donnent le même résultat :

```
>>> liste[1:-2]
[7, 11, 14]
>>> liste[-5:4]
[7, 11, 14]
```

Question : quelles sous-listes obtient-on avec les syntaxes `liste[2:5]`, `liste[2:-1]`, `liste[-4:5]` et `liste[-4:-1]` ?
Réponse : 4 fois la même sous-liste `[11, 14, 3]` car ces 4 écritures correspondent au même tranchage :

```
>>> liste[2:5]
[11, 14, 3]
>>> liste[2:-1]
[11, 14, 3]
>>> liste[-4:5]
[11, 14, 3]
>>> liste[-4:-1]
[11, 14, 3]
```

Dans l'indice écrit entre crochet, si le second index est égal au premier ou est à gauche du premier dans le découpage en tranches illustré ci-dessus alors la sous-liste sélectionnée sera **la liste vide**, qui s'écrit [] et qui ne contient aucun élément. Exemples :

```
>>> liste[5:2]
[]
>>> liste[4:4]
[]
>>> liste[-2:-5]
[]
>>> liste[6:-4]
[]
>>> liste[-3:1]
[]
```

Si on ajoute un troisième chiffre dans l'indice entre crochet il correspond **au "pas"** et permet de sélectionner par exemple un élément sur 2 ou un élément sur 3. Le pas peut être négatif. Exemples :

```
>>> liste[0:6:2]
[18, 11, 3]
>>> liste[1:6:2]
[7, 14, 9]
>>> liste[0:6:3]
[18, 14]
>>> liste[6:1:-3]
[9, 11]
>>> liste[::-1]
[9, 3, 14, 11, 7, 18]
```

Un indice composé d'un chiffre suivi de deux-point extrait une sous-liste allant de l'index indiqué **jusqu'à la fin** de la liste. Exemples :

```
>>> liste[0:]
[18, 7, 11, 14, 3, 9]
>>> liste[1:]
[7, 11, 14, 3, 9]
>>> liste[3:]
[14, 3, 9]
>>> liste[4:]
[3, 9]
```

Un indice composé de deux-point suivi d'un chiffre extrait une sous-liste **allant du début** de la liste jusqu'à l'index indiqué. Exemples :

```
>>> liste[:6]
[18, 7, 11, 14, 3, 9]
>>> liste[:5]
[18, 7, 11, 14, 3]
>>> liste[:3]
[18, 7, 11]
>>> liste[:1]
```

Enfin, un indice composé seulement du caractère deux-point (sans indexs chiffrés) sélectionne **la liste entière** :

```
>>> liste[:]
[18, 7, 11, 14, 3, 9]
```

La technique de tranchage permet d'extraire une sous-liste d'une liste en utilisant une syntaxe simple :

Extraction d'une sous-liste par tranchage	
syntaxe	**éléments sélectionnés**
liste[n]	extrait seulement l'élément d'indice n (pas une sous-liste)
liste[n:m]	extrait la sous-liste composée des éléments compris entre les indexs n et m
liste[n:m:k]	extrait la sous-liste composée des éléments compris entre les indexs n et m par pas de k
liste[n:]	tous les éléments de n jusqu'à la fin de la liste
liste[:m]	tous les éléments du début de la liste jusqu'à m
liste[:]	la liste entière
liste[::k]	un élément sur k dans la liste entière

III – Création d'une liste numérique avec la fonction range()

La fonction **range()** permet de créer facilement des listes numériques. ***Attention :*** la fonction **range()** ne renvoie pas directement une liste. Pour obtenir un objet de type liste il faut le convertir grâce à la fonction **list** qui convertit en liste tout objet. La fonction **range()** permet de créer rapidement des listes d'entiers.

Avec un seul paramètre **range(n)** crée une liste numérique contenant les entiers de 0 à n-1 (par pas de 1) :

```
>>> list(range(20))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Avec 2 paramètres **range(n,m)** crée une liste numérique contenant les entiers de n à m-1 (par pas de 1) :

```
>>> list(range(20,29))
[20, 21, 22, 23, 24, 25, 26, 27, 28]
```

Avec 3 paramètres **range(n,m,k)** crée une liste numérique contenant les entiers de n à m-1 **par pas de k** :

```
>>> list(range(20,29,3))
[20, 23, 26]
```

IV – Création d'une liste en compréhension

En Python on peut créer rapidement des listes par une instruction du type **[valeur boucle]**. On parle alors de listes **définies par compréhension**. Les compréhensions de liste en Python sont des outils puissants qui permettent de créer et de manipuler des listes de manière concise et lisible. Exemples :

`liste = [i for i in range(8)]` donnera la liste des entiers de 0 à 7 : `[0, 1, 2, 3, 4, 5, 6, 7]`

`liste = [2*i for i in range(8)]` donnera la liste suivante :`[0, 2, 4, 6, 8, 10, 12, 14]`

`liste = [i for i in range(30) if i%2==0]` donnera la liste des nombres pairs compris entre 0 et 29 : `[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]`

`liste = [i for i in range(30) if i%2!=0 and i>10]` donnera la liste des nombres impairs supérieurs à 10 entre 0 et 29 : `[11, 13, 15, 17, 19, 21, 23, 25, 27, 29]`

`liste=[i for i in range(30) if i%2!=0 and i>15 or i<8]` donnera la liste des nombres impairs supérieurs à 15, ou des nombres inférieurs à 8, dans l'intervalle des entiers allant de 0 à 29 (le ET est prioritaire devant le OU) : `[0, 1, 2, 3, 4, 5, 6, 7, 17, 19, 21, 23, 25, 27, 29]`

`liste=[i for i in range(30) if i%2!=0 and (i>15 or i<8)]` donnera la liste des nombres impairs soit supérieurs à 15 soit inférieurs à 8, dans l'intervalle des entiers allant de 0 à 29 (les parenthèses priorise ici le OU devant le ET) : `[1, 3, 5, 7, 17, 19, 21, 23, 25, 27, 29]`

Grâce à la définition de liste en compréhension, il est possible de générer toute sorte de listes complexes (numériques ou non numériques) en une seule ligne de code. Exemple de génération d'une liste non numérique en compréhension :

`liste=[3*i for i in "ABCDEF" if i!='E']` donnera la liste `['AAA', 'BBB', 'CCC', 'DDD', 'FFF']`

V – Manipulation des listes en Python

Il existe 3 opérateurs applicables sur des listes en Python, les voici :

Les opérateurs applicables sur une liste	
opérateur	**opération effectuée**
+	concaténation de deux listes
*****	duplication d'une liste
in	test d'appartenance à une liste : le résultat est Booléen (soit **True** soit **False**)

Exemples : `[1,2]+[3,4,5]` renvoie la liste `[1, 2, 3, 4, 5]`
`['oui',78,'A']+[3.14,'XYZ']+[2]` renvoie la liste `['oui', 78, 'A', 3.14, 'XYZ', 2]`
`[5]*3` renvoie la liste `[5, 5, 5]`
`['K',8]*2+[7]*5` renvoie la liste `['K', 8, 'K', 8, 7, 7, 7, 7, 7]`
`'G' in ['E','F','G',4,5,6]` renvoie `True`
`7 in ['E','F','G',4,5,6]` renvoie `False`

Voici les fonctions Python applicables à une liste :

Les fonctions externes applicables à une liste	
fonction	**opération effectuée**
len	renvoie la longueur de la liste (nombre d'éléments)
del	supprime un ou plusieurs éléments dans la liste
sorted	renvoie la liste avec les éléments rangés dans l'ordre croissant (trie une liste)
shuffle	mélange une liste en place (importer le module random : `from random import *`)

Exemples : `sorted([8,5,4,7,3])` renvoie `[3, 4, 5, 7, 8]` `len([8,5,4,7,3])` renvoie 5
Après `liste=[7,2,5];shuffle(liste)` la liste est mélangée `del liste[1]` supprime l'élément 1 de la liste

VI – Les méthodes appliquées à un objet liste

Il existe 9 méthodes applicables à un objet liste en Python, les voici :

Les méthodes propres à une liste	
méthode	**opération effectuée**
append	ajoute un nouvel élément à la fin de la liste
count	renvoie le nombre d'occurrence d'un élément dans la liste
extend	allonge la liste avec une autre liste
reverse	inverse l'ordre des éléments dans la liste
sort	ordonne les éléments de la liste dans l'ordre croissant
insert	ajoute un nouvel élément à une position particulière
remove	supprime un élément de la liste (le premier trouvé en cas de doublons)
index	renvoie l'index d'un élément (le premier trouvé en cas de doublons)
pop	renvoie le dernier élément et le supprime de la liste

Exemples de modification d'une liste valant à chaque fois `liste=[4,2,9,1,7]` avant l'instruction :

exemple d'instruction	état de la liste après l'exécution de cette instruction
`liste.append(83)`	`[4,2,9,1,7,83]`
`liste.extend([15,47,96])`	`[4,2,9,1,7,15,47,96]`
`liste.reverse()`	`[7,1,9,2,4]`
`liste.sort()`	`[1, 2, 4,7,9]`
`liste.insert(3,58)`	`[4, 2, 9,58,1,7]`
`liste.pop()`	`[4,2,9,1]`

Exemples d'information obtenue sur une liste valant à chaque fois `liste=[3,8,5,6,8,4,4]` avant l'instruction :

exemple d'instruction	valeur renvoyée par cette instruction
`liste.count(5)`	1
`liste.count(8)`	2
`liste.count(7)`	0
`liste.index(3)`	0
`liste.index(8)`	1
`liste.index(4)`	5
`liste.pop()`	4

VII – Les listes de listes

Une liste de listes permet de représenter un **tableau** à 2 dimensions. Prenons par exemple ce tableau à 9 éléments (il possède 3 lignes et 3 colonnes) :

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

En Python il se code comme une liste de listes avec `tableau=[[1,2,3],[4,5,6],[7,8,9]]` `tableau[0]` est la première ligne, soit `[1, 2, 3]`, `tableau[1]` est la deuxième ligne, soit `[4, 5, 6]` et `tableau[2]` est la troisième ligne, soit `[7, 8, 9]`.

Pour accéder à un seul élément précis du tableau il faut utiliser **2 indices** à la fois : le premier indique la **ligne** (entre 0 et 2) et le second indique la **colonne** (entre 0 et 2 pour notre exemple). Exemples :

`tableau[0][0]` renvoie **1**	`tableau[0][1]` renvoie **2**	`tableau[0][2]` renvoie **3**
`tableau[1][0]` renvoie **4**	`tableau[1][1]` renvoie **5**	`tableau[1][2]` renvoie **6**
`tableau[2][0]` renvoie **7**	`tableau[2][1]` renvoie **8**	`tableau[2][2]` renvoie **9**

Construisons maintenant un tableau nommé `tab` à 12 éléments : 4 lignes et 3 colonnes. On commence par un tableau vide à 4 lignes : `tab=[[],[],[],[]]`. Nous allons remplir progressivement ce tableau vide, ligne par ligne. Remplissons la première ligne : `tab[0]=[7,5,3]`. Remplissons la deuxième ligne élément par élément : `tab[1].append(23)`, `tab[1].append(54)` et `tab[1].append(17)`. Mettons 3 fois le même élément dans la troisième ligne : `tab[2]=3*[8]`. Et finissons par la dernière ligne : `tab[3]=list("ABC")`

7	**5**	**3**
23	**54**	**17**
8	**8**	**8**
A	**B**	**C**

On obtient alors le tableau ci-contre de dimensions 4x3 (4 lignes et 3 colonnes) qui se code comme ceci en Python :

`[[7, 5, 3], [23, 54, 17], [8, 8, 8], ['A', 'B', 'C']]`