1 NSI _ Chapitre 11 complémént sur les listes.docx

1. Listes vides

```
#création d'une liste vide
ma_liste = []
#équivalent
ma_liste = list()
```

2. Listes avec compréhension

Une liste en compréhension est une expression littérale de liste équivalente à une boucle for qui construirait la même liste en utilisant la méthode append ()

#listes définies en compréhension

A=[0 **for** i **in** range(5)]	A=list() for i in range(5): A.append(0)
print(A)	

#donne [0, 0, 0, 0, 0]

```
B=[x**2 for x in range(11)]
print(B)
#donne [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

C=[x for x in range(25) if x%3==0]
print(C)
#donne [0, 3, 6, 9, 12, 15, 18, 21, 24]
```

3. Listes à deux dimensions

On peut la créer de façon exhaustive :

```
D=[[0,1,2],[3,4,5],[6,7,8]]
print(D)
print("D[0][0] : {}".format(D[0][0]))
print("D[0][2] : {}".format(D[0][2]))
print("D[2][0] : {}".format(D[2][0]))
```

```
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
D[0][0] : 0
D[0][2] : 2
D[2][0] : 6
```

On peut aussi utiliser les boucles :

```
nb_lin = 3
nb_col = 3
```

```
mat = []
for i in range(nb_lin):
    row = []
    for j in range(nb_col):
        elt = int(input("Donner M {}/{}\t".format(i,j)))
        row.append(elt)
    mat.append(row)
print(mat)
```

```
Donner M 0/0    0
Donner M 0/1    1
Donner M 0/2    2
Donner M 1/0    3
Donner M 1/1    4
Donner M 1/2    5
Donner M 2/0    6
Donner M 2/1    7
Donner M 2/2    8
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

On peut utiliser des générateurs

```
A =[[i for i in range(3)],[3+j for j in range (3)],[6+j for j in range (3)]]
print(A)
print("A[0][0] : {}".format(A[0][0]))
print("A[0][2] : {}".format(A[0][2]))
print("A[2][0] : {}".format(A[1][0]))
```

```
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
A[0][0] : 0
A[0][2] : 2
A[2][0] : 6
```

On peut utiliser l'écriture par compréhension

```
A =[[i + 3*j for i in range(3)] for j in range(3)]
```

```
>>> [[i+3*j for i in range(3)] for j in range(3)]
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

4. Découpage en tranches (troncature)

Le terme de tranche (slice en anglais) est associé à l'idée de découpage.

En PYTHON, une tranche permet le découpage de structures de données séquentielles, typiquement des chaînes de caractères, des listes. Ce sont des expressions qui permettent d'extraire des éléments d'une séquence en une ligne de code. Leur intérêt réside essentiellement dans la concision et la souplesse de leur syntaxe.

4.1. Tranches avec deux indices

```
>>> alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

>>> alpha[4:16]
'EFGHIJKLMNOP'
```

On interprète avec les indices de la liste :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
c	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
-i	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

ici, alpha[4:16] est une tranche et a pour valeur la chaîne extraite de la chaîne alpha dont les éléments sont situés :

• à droite de l'élément d'indice 4 ;

• strictement à gauche de l'élément d'indice 16, autrement dit jusqu'à l'indice 15 inclus.

```
>>> alpha[5:-3]
'FGHIJKLMNOPQRSTUVW'
```

ici, alpha[5:-3] est une tranche et a pour valeur la chaîne extraite de la chaîne alpha dont les éléments sont situés :

• à droite de l'élément d'indice 5 soit « F »;

• strictement à gauche de l'élément d'indice -3, autrement dit jusqu'à « X » non inclus donc « W ».

4.2. Omission d'indices

```
>>> alpha[:5] # Les 5 premiers
'ABCDE'

>>> alpha[-5:] # Les 5 derniers
'VWXYZ'

>>> alpha[5:] # Tous sauf les 5 premiers
'FGHIJKLMNOPQRSTUVWXYZ'

>>> alpha[:-5] # Tous sauf les 5 derniers
'ABCDEFGHIJKLMNOPQRSTU'

>>> alpha[:] # Toute la liste
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

La tranche L[:] permet donc d'obtenir une copie d'une liste L, tout comme list(L)

4.3. Tranches étendues

Comme pour range, la syntaxe des tranches admet un 3e paramètre : un pas (step en anglais).

```
>>> alpha[4:23:3]
'EHKNQTW'
```

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
c	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

ici, alpha[4:23 :3] est une tranche et a pour valeur la chaîne extraite de la chaîne alpha dont les éléments sont situés :

- à droite de l'élément d'indice 4 ;
- strictement à gauche de l'élément d'indice 23, autrement dit jusqu'à l'indice 22 inclus soit « W ».
- par pas de trois soit E,H,K,N,Q,T,W

Idem par absence d'indice

```
>>> alpha[::-5] # De 5 en 5 à partir de la fin
'ZUPKFA'
```

Cas particulier inversion de liste

```
>>> alpha[::-1]
'ZYXWVUTSRQPONMLKJIHGFEDCBA'
```