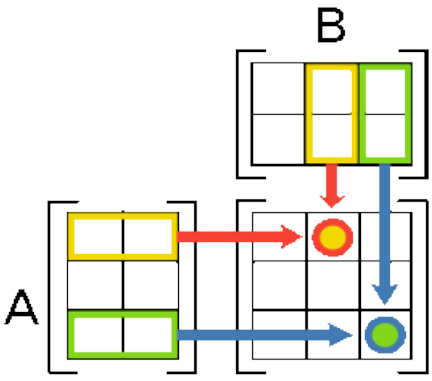
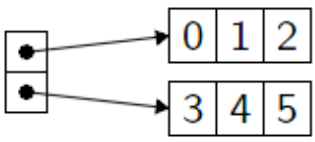
Listes de listes - Matrices




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

1	2	3
4	5	6



Listes de listes - matrices

1. Présentation des matrices

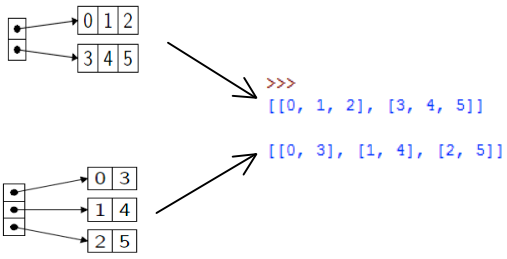
On peut choisir de représenter une matrice de dimensions (n,p) par ***un tableau de longueur n, dont les éléments sont des tableaux de longueur p***. Ainsi la matrice (2,3) peut être définie en Python:

$$M = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{pmatrix}$$

```
m=[[0,1,2],[3,4,5]]
print(m)

print()

m=[[0,3],[1,4],[2,5]]
print(m)
```



Une matrice est tout simplement une ***liste de listes.***

2. Accès aux éléments des matrices

On accède à l'élément ***Mi,j*** avec l'expression **`m[i][j]`.**

```
m=[[0,1,2],[3,4,5]]
print(m[0])
print(m[0][1])

print()

m=[[0,3],[1,4],[2,5]]
print(m[0])
print(m[0][1])
```

```
>>>
[0, 1, 2]
1

[0, 3]
3
>>>
```

3. Partage de tableau – Création par copie

ATTENTION **Partage de tableau**

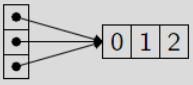
Supposons que l'on veuille construire maintenant la matrice

$$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix}$$

On pourrait être tenté de construire un tableau `v = [0, 1, 2]`, puis de l'utiliser trois fois pour chacune des lignes de la matrice :

```
v = [0, 1, 2]
m = [v, v, v]
```

Il s'agit bien là d'une matrice de dimensions $(3, 3)$. Cependant, sa représentation en mémoire n'est pas la même que dans l'exemple ci-dessus, et montre au contraire un *partage* du tableau `v` entre les trois lignes.

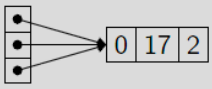


En particulier, si on affecte un élément de la matrice, par exemple l'élément `m[0][1]` avec l'instruction

```
m[0][1] = 17
```

c'est en fait toute la colonne, c'est-à-dire les trois éléments `m[0][1]`, `m[1][1]`, `m[2][1]`, qui sont modifiés :

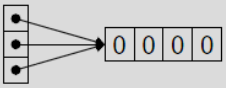
Listes de listes - matrices



On peut l'observer facilement avec `print(m)`, qui affiche

```
[[0, 17, 2], [0, 17, 2], [0, 17, 2]]
```

Pour la même raison, on ne peut pas utiliser l'expression `[[0] * 4] * 3` pour créer une matrice de dimensions $(4, 3)$ initialisée avec des zéros, car elle correspond en fait à la situation suivante :



Programme python pour tester ce problème de copie de matrices :

```
v=[0,1,2]
m=[v,v,v]
print(m)

m[0][1]=17
print(m)

print()

u=[[0]*4]*3
print(u)

u[0][1]=12
print(u)
```

```
>>>
[[0, 1, 2], [0, 1, 2], [0, 1, 2]]
[[0, 17, 2], [0, 17, 2], [0, 17, 2]]

[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 12, 0, 0], [0, 12, 0, 0], [0, 12, 0, 0]]
>>>
```

Listes de listes - matrices

4. Création de matrices

Pour créer une matrice de grande taille, on ne souhaite évidemment pas donner tous ses éléments explicitement. Par ailleurs, les dimensions peuvent être contenues dans des variables. Ecrivons donc une fonction `creer_matrice` pour construire une matrice M de dimensions (n,p) où chaque élément Mi,j est initialisé avec une valeur v.

On procède en créant un tableau de taille n initialisé avec `None`, puis on affecte à chacune de ses cases un tableau de taille p différent :

```
import time
from time import clock
```

Module `time` pour ***calculer temps exécution***

```
def creer_matrice(n,p,v):
    global m
    m=[None]*n
    for i in range(n):
        m[i]=[v]*p
    return m

#main
n=eval(input("entrez le nombre de lignes: "))
p=eval(input("entrez le nombre de colonnes: "))
v=eval(input("entrez la valeur initiale pour chaque element: "))

print(creer_matrice(n,p,v))
```

ici `[v]*p` ne créé pas alias (comme p7) car v n'est pas de type liste !!

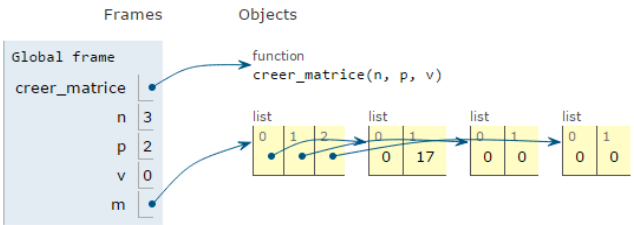
```
print()
tic = time.clock()
creer_matrice(n,p,v)
tac = time.clock()

print("Duree: ", tac-tic)

print()

m[0][1]=17
print(m)
print()
```

Instructions pour ***calculer temps exécution***



Résultats :

```
>>>
entrez le nombre de lignes: 4
entrez le nombre de colonnes: 5
entrez la valeur initiale pour chaque element: 0
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]

Duree:  7.865937072503421e-06

[[0, 17, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Autre solution avec append :

```
import time
from time import clock

def creer_matrice(n,p,v):
    m=[]
    for i in range(n):
        ligne=[v]*p
        m.append(ligne)
    return m

#main
n=eval(input("entrez le nombre de lignes: "))
p=eval(input("entrez le nombre de colonnes: "))
v=eval(input("entrez la valeur initiale pour chaque element: "))

print(creer_matrice(n,p,v))

print()

tic = time.clock()
creer_matrice(n,p,v)
tac = time.clock()

print("Duree: ", tac-tic)
```

```
>>>
entrez le nombre de lignes: 4
entrez le nombre de colonnes: 5
entrez la valeur initiale pour chaque element: 0
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]

Duree:  1.3679890560875514e-05
```

Autre solution plus performante avec ***listes en compréhension*** :

```
import time
from time import clock

def creer_matrice(n,p,v):
    return [[v]*p for i in range(n)]

#main
n=eval(input("entrez le nombre de lignes: "))
p=eval(input("entrez le nombre de colonnes: "))
v=eval(input("entrez la valeur initiale pour chaque element: "))

print(creer_matrice(n,p,v))

print()

tic = time.clock()
creer_matrice(n,p,v)
tac = time.clock()

print("Duree: ", tac-tic)
```

```
>>>
entrez le nombre de lignes: 4
entrez le nombre de colonnes: 5
entrez la valeur initiale pour chaque element: 0
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]

Duree:  8.891928864569084e-06
```

5. Copie de matrices – Matrice CARREE

Si on veut obtenir une ***matrice CARREE*** m de dimensions (n,p), par copie de la première ligne, il faut prendre soin de copier chacune de ses n lignes, pour obtenir autant de nouveaux tableaux. De manière élémentaire, on commence par construire un tableau r de taille n initialisé à `None`. Puis on affecte chaque ligne `r[i]` avec une copie de la ligne `m[i]` obtenue en utilisant la notation `m[:]`.

Listes de listes - matrices

```
def copie_matrice(m):
    n=len(m)
    global r
    r=[None]*n
    for i in range(n):
        r[i]=m[:]
    return r

#main
m=eval(input("entrez ligne a  copier:  "))
print(copie_matrice(m))

print(r[0])
print(r[0][2])
r[0][1]=22
print(r)
```

```
>>>
entrez ligne a  copier:  [5,6,7]
[[5, 6, 7], [5, 6, 7], [5, 6, 7]]
[5, 6, 7]
7
[[5, 22, 7], [5, 6, 7], [5, 6, 7]]
>>>
```

Autre solution avec listes en compréhension :

```
def copie_matrice(m):
    return [m[:] for i in range(len(m))]

#main
m=eval(input("entrez ligne a  copier:  "))
print(copie_matrice(m))
t=copie_matrice(m)

print(t[0][1])
```

```
>>>
entrez ligne a  copier:  [5,6,7]
[[5, 6, 7], [5, 6, 7], [5, 6, 7]]
Traceback (most recent call last):
  File "I:\2013 2014 PTSI\INFORMATIQUE\INFO PTSI 2014-2015\S17-18. Matrices et t
ableaux\Cours\16. copie de matrice1 compréhension.py", line 9, in <module>
    print(m[0][1])
TypeError: 'int' object is not subscriptable
```

ATTENTION Les tableaux Python

Python propose plusieurs structures de tableaux, chacune prévue pour un usage spécifique. Dans ce chapitre, nous avons utilisé les « listes », dont on verra au chapitre qu'elles sont en fait un peu plus que de simples tableaux.

Il existe également une bibliothèque appelée `array` qui, comme son nom l'indique, correspond effectivement à une structure de tableaux. Mais il s'agit uniquement d'une représentation plus compacte pour des tableaux très spécifiques, dont les éléments sont tous d'un même type numérique simple (caractère, entier ou flottant). En particulier, nous ne pourrions pas utiliser la bibliothèque `array` pour représenter des matrices.

Enfin la bibliothèque `numpy` propose aussi un type *ndarray* qui permet de représenter des tableaux de dimension arbitraire dont tous les éléments sont d'un même type, principalement utilisés pour le calcul matriciel. Attention à ne pas confondre ce type avec le précédent, même si ses valeurs s'affichent parfois aussi sous la forme `array(...)`.