



# Matrices C#

---

NORA IZRI – DÉPARTEMENT IBO

# Exercices sur schooding

---

# Exercice 1

---

Ecrire la méthode ***bool Modif1OccurrValeur(int[] tableau, int val, int modif)*** qui permet de modifier uniquement la première occurrence de l'entier « val » par l'entier « modif » dans le tableau en entrée. Si « val » a été modifiée, la méthode retourne true, sinon false.

Soit le tableau suivant :

1	2	3	4	2	6	2
---	---	---	---	---	---	---

val=2 et modif=45

Après appel à la méthode, elle retourne true et le tableau initial devient :

1	45	3	4	2	6	2
---	----	---	---	---	---	---

# Exercice 2

---

Ecrire la méthode **bool EstPresentMot(string[] tab, string mot)** qui permet vérifier si le "mot" est présent dans le tableau de string en entrée ou pas.

# Exercice 3

---

Ecrire la méthode ***void InverserTableau(int[] tableau)*** qui permet d'inverser les éléments se trouvant dans le tableau en paramètre d'entrée

Soit le tableau suivant :

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Après appel à la méthode Exercice1Tableau, le tableau initial devient :

7	6	5	4	3	2	1
---	---	---	---	---	---	---

# Matrices

---

# Déclaration

## Syntaxe

**type[ , ] nomDeLaMatrice2D;**

**type[ , , ] nomDeLaMatrice3D;**

Etc. pour N dimensions

Ceci permet juste de déclarer la matrice, elle n'existe toujours pas ➔ la valeur est **null**.

### Code

```
1 int[,] matriceEntiers; // déclaration d'une matrice à 2D,
   qui pourra contenir des entiers
2
3 double[, ,] matriceReels = null; // déclaration d'une
   matrice à 3D, qui pourra contenir des réels ; ici la
   valeur null est explicitement renseignée.
```

# Matrice - Allocation mémoire

Syntaxe

```
nomDeLaMatrice2D = new type[tailleDim1, tailleDim2];
```

```
int[,] matrice = null; //déclaration  
matrice = new int[2, 3]; //allocation mémoire de 6 cases, sur 2 lignes et 3  
colonnes, pouvant chacune contenir une valeur entière
```

Formule compacte ➔ création d'une matrice : déclaration + allocation mémoire

Syntaxe

```
type[ , ] nomDeLaMatrice2D = new type[tailleDim1, tailleDim2];
```

```
int[ , ] matriceEntiers= new int[2 , 3];  
int[ , ] matriceNegatifs= new int[5 , 8];
```



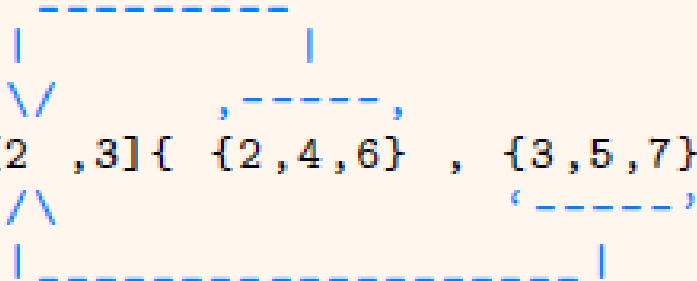
# Création Matrice avec initialisation

## Code

```
int[,] matrice = new int[2,3] { {2,4,6} , {3,5,7} };
```

## Code (idem)

```
//  
//  
//  
int[,] matrice = new int[2 ,3]{ {2,4,6} , {3,5,7} };  
//  
//
```



## Code (version simplifiée)

```
int[,] matrice = new int[,] { {2,4,6} , {3,5,7} };
```

## Code (version encore plus simplifiée)

```
int[,] matrice = { {2,4,6} , {3,5,7} };
```

# Nombre d'éléments et dimensions

- Nombre d'éléments/cases → propriétés Length
- Taille de chaque dimension → méthode **GetLength** (n°Dimension)
  - les dimensions sont identifiées de **0** à N-1

Exemple : `int[,] matrice = new int[2, 3];` //création d'une matrice d'entiers int de taille 6 cases sur 2 lignes et 3 colonnes

```
Console.WriteLine("Nombre de lignes : " + matrice.GetLength(0));  
Console.WriteLine("Nombre de colonnes : " + matrice.GetLength(1));  
Console.WriteLine("Nombre de cases : " + matrice.Length);
```

```
Nombre de lignes : 2  
Nombre de colonnes : 3  
Nombre de cases : 6
```

# Accès aux éléments d'une matrice

- Accès en **lecture** ou en **écriture** à une case

```
nomDeLaMatrice2D [indexDim1, indexDim2];
```

- Exemple `int[,] matrice = new int[2, 3];` //création d'une matrice d'entiers int de taille 6 cases sur 2 lignes et 3 colonnes

```
matrice[0, 0] = 5; // accès en écriture à l'élément (0,0)
```

```
Console.WriteLine(matrice[0, 0]); //accès en lecture à l'élément (0,0)
```

# Exercices

---

# Exercice 1

---

Ecrire la méthode void Exercice1Matrice() qui permet de :

1. Créer la matrice 2D de dimensions 3x4 suivante :

1	4	7	10
2	5	8	11
3	6	9	12

2. Afficher la matrice 2D en la parcourant ligne par ligne puis colonne par colonne (pour chaque ligne parcourir toutes les colonnes).

Pour la matrice ci-dessus, par exemple, l'affichage attendu :

1 4 7 10

2 5 8 11

3 6 9 12

## Exercice 2

---

1. Ecrire une méthode ***bool RechercheMatrice(int[,] mat, int val)*** qui retourne « true » si la valeur « val » existe dans la matrice « mat », « false » sinon.
2. Ecrire une méthode ***void TestRechercheMat()*** qui permet de :
  1. Déclarer une matrice d'entiers (dimensions et remplissage de votre choix : en dur ou par saisie utilisateur)
  2. Appeler la méthode « RechercheMatrice » et affiche un message significatif à l'utilisateur

# Exercice 3

---

1. Ecrire une méthode ***bool TriangulaireSuperieure(int[,] matrice)*** qui retourne true si la matrice est triangulaire supérieure (matrice carrée dont toutes les valeurs sous la diagonale principale sont 0), false sinon. Par exemple, la matrice { {1 ,2 ,0} , {0 ,4 ,5}, {0,0, 6} } est diagonale supérieure alors que {{1,2,3}, {0,4,5}} ou { {1 ,2 ,0} , {0 ,4 ,5}, {3,0, 6} } ne le sont pas. On suppose que matrice n'est pas nulle ou vide.
2. Ecrire une méthode TestTriangulaireSuperieure()

# Exercice 4

---

1. Ecrire une méthode ***int SommeColonneMatrice(int[,] matrice, int col)*** qui retourne la somme des entiers se trouvant dans la colonne « col » si cela est possible sinon retourne -1.
2. Ecrire une méthode ***void TestSommeColonne()*** qui permet de :
  1. Déclarer une matrice d'entiers (dimensions et remplissage de votre choix : en dur ou par saisie utilisateur)
  2. Appeler la méthode « SommeColonneMatrice » et affiche un message significatif à l'utilisateur



# Exercice 5

---

1. Ecrire une méthode ***double[] ElementMat(double[,] matrice)*** qui prend en entrée une matrice de réels et retourne un tableau de réels contenant tous les éléments de la matrice.
2. Ecrire une méthode ***void TestElementMat()*** qui permet de :
  1. Déclarer une matrice de réels (dimensions et remplissage de votre choix : en dur ou par saisie utilisateur)
  2. Appeler la méthode « ElementMat »
  3. Parcourir et afficher les éléments du tableau en sortie

# Tableaux de tableaux

---

# Tableaux de tableaux

---

- La plupart des langages de programmation ne disposent pas de cette notion de matrice.
- Manipulation de tableaux de tableaux → Chaque élément d'un tableau soit lui-même un tableau ... et ainsi de suite.

# Exemple > Tableaux de tableaux

## Code (Déclaration)

```
1 int [][] tab2D;  
2 // <=> int [][] tab2D = null;
```

## Code (Allocation mémoire du tableau principal)

```
1 tab2D = new int [2] [];
```

## Code (Création du 1<sup>er</sup> élément (*i.e.* du 1<sup>er</sup> tableau imbriqué)

```
1 tab2D [0] = new int [3];  
2  
3 tab2D [0] [0] = 10;  
4 tab2D [0] [1] = 20;  
5 tab2D [0] [2] = 30;
```

## Code (Création du 2<sup>nd</sup> élément (*i.e.* du 2<sup>nd</sup> tableau imbriqué)

```
1 tab2D [1] = new int [4] {40, 50, 60, 70};
```

10	20	30	
40	50	60	70

10	20	30	
40	50	60	70