

# C#

## ▼ Les variables

### ▼ initialisation

- soit : `type_variable v = initial ;`
- ou `var v = valeur ;` dans ce cas initialisation obligatoire pour prendre le type du valeur , et v ne peut pas change sa type .l.
- `const v = jhjh;`

### ▼ Conversion chaîne de caractères <-> nombre

`nombre → chaîne : nombre.ToString()`

`chaîne → int : int.Parse(chaine)`

`chaîne → long : long.Parse(chaine)`

`chaîne → double : double.Parse(chaîne)`

`chaîne → float : float.Parse(chaîne)`

### ▼ note

tout est objet pas de type primitive .

- Afficher sur la console : `Console.WriteLine(expression)` ou `Console.Write (expression)`
- lecture : `string ligne = Console.ReadLine();`
- la boucle foreache est une boucle pour lecture seulement .

### ▼ compare two strings

#### ▼ equals

return bool .

#### ▼ compareTo

return int

### ▼ Tableau

- Déclaration : `Type[] tableau=new Type[n]`
- taille est `length`

- initial tab = {0,2,15,} ou string [] jours = new string [] { " Lundi ", " Mardi ", " Mercredi ", " Jeudi ", " Vendredi ", " Samedi ", " Dimanche " };

#### ▼ tableau plusieurs dimension

- Type[,] tableau=new Type[n,m]; 2dimension
- tableau.GetLength(0) : return les length du tab 0
- What are the differences between multidimensional arrays `double[,]` and array-of-arrays `double[][]` : is in the jaged we can

```
int[][] jagged = new int[3][];
jagged[0] = new int[4] { 1, 2, 3, 4 };
jagged[1] = new int[2] { 11, 12 };
jagged[2] = new int[3] { 21, 22, 23 };
```

#### ▼ les collection

##### ▼ geniric

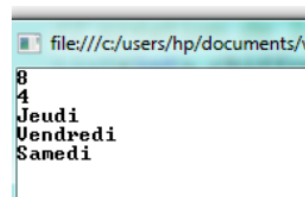
t List<T>

```
// création de la liste
List<int> chiffres = new List<int>();
chiffres.Add(8); // chiffres contient 8
chiffres.Add(9); // chiffres contient 8, 9
chiffres.Add(4); // chiffres contient 8, 9, 4
chiffres.RemoveAt(1); // chiffres contient 8, 4

foreach (int chiffre in chiffres)
{
    Console.WriteLine(chiffre);
}

List<string> jours = new List<string>();
jours.Add("Jeudi");
jours.Add("Vendredi");
jours.Add("Samedi");

foreach (string chiffre in jours)
{
    Console.WriteLine(chiffre);
}
```



```
file:///c:/users/hp/documents/\
8
4
Jeudi
Vendredi
Samedi
```

- public int Count {get;} □ nombre d'éléments de la liste
- public void Add(T item) □ ajoute item à la liste
- public int BinarySearch<T>(T item) □ rend la position de item dans la liste s'il s'y trouve sinon un nombre <0

- `public void Clear()` □ supprime tous les éléments de la liste
- `public bool Contains(T item)` □ rend `True` si `item` est dans la liste, `False` sinon
- `public void CopyTo(T[] tableau)` □ copie les éléments de la liste dans `tableau`.
- `public int IndexOf(T item)` □ rend la position de `item` dans `tableau` ou `-1` si valeur n'est pas trouvée.
- `public void Insert(T item, int index)` □ insère `item` à la position `index` de la liste
- `public bool Remove(T item)` □ supprime `item` de la liste. Rend `True` si l'opération réussit, `False` sinon.
- `public void RemoveAt(int index)` □ supprime l'élément n° `index` de la liste
- `public void Sort()` □ trie la liste selon l'ordre défini par le type des éléments de la liste
- `public T[] ToArray()` □ rend les éléments de la liste sous forme de tableau

#### ▼ dictionnaire

Création : `Dictionary<TKey,TValue> D=new Dictionary<TKey,TValue>();`

Accès à la valeur associée à la clé `C` dans le dictionnaire `D` : `D[C]`.

Ajouter une valeur dans le dictionnaire : `D[Cle1] = Val1`.

#### ▼ ses methodes

`public int Count {get;}` □ taille de la collection

`public void Add(TKey key, TValue value)` □ ajoute le couple (`key`, `value`)

au dictionnaire

`public void Clear()` □ supprime tous les couples du dictionnaire

`public bool ContainsKey (TKey key)` □ rend `True` si `key` est une clé du dictionnaire, `False` sinon

public bool ContainsValue (TValue value) □ rend True si  
 value est une valeur du dictionnaire, False sinon  
 public void CopyTo(T[] tableau) □ copie les éléments de la  
 liste dans tableau.  
 public bool Remove(TKey key) □ supprime du dictionnaire le  
 couple de clé key. Rend True si l'opération réussit,  
 False sinon

```
// création d'un dictionnaire <int,string> (code /ville)
Dictionary<int, string> villeCode = new Dictionary<int, string>();
villeCode[50] = "NADOR";
villeCode[1] = "RABAT";
villeCode[45] = "AL HOCEIMA";
// nbre d'éléments dans le dictionnaire
Console.WriteLine("Le dictionnaire a " + villeCode.Count + " éléments");
Console.WriteLine("Les villes avec leurs codes :");
foreach (int code in villeCode.Keys)
{
    Console.WriteLine( code + " \t " + villeCode[code]);
}
```

▼ non genbiric

ArrayList

▼ Les énumérations

enum Mentions { Passable, Assez\_Bien, Bien, Très\_Bien, Excellent };

```
// une variable qui prend ses valeurs dans l'énumération Me
Mentions maMention = Mentions.Passable;
// affichage valeur variable
Console.WriteLine("mention=" + maMention);
// test avec valeur de l'énumération
if (maMention == Mentions.Passable)
{
    Console.WriteLine("Vous pouvez faire mieux ...");
}
// liste des mentions sous forme de chaînes
foreach (Mentions m in Enum.GetValues(maMention.GetType()))
{
    Console.WriteLine(m);
}
//liste des mentions sous forme d'entiers
foreach (int m in Enum.GetValues(typeof(Mentions)))
{
    Console.WriteLine(m);
}
```

## ▼ Passage par référence avec le mot clé ref

```
public void changeAge(ref int page)
{
    page = 12;
    this.age = page;
}
//Méthode de test
public static void Main(string[] args)
{
    int agetest = 36;
    Etudiant et1 = new Etudiant("El Hani", "Karima", 31);
    et1.afficheEtudiant();
    et1.changeAge(ref agetest);
    Console.WriteLine("\nAprès la modification :");
    et1.afficheEtudiant();
    Console.WriteLine("agetest=" + agetest);
    Console.Read();
}
```

Passage par référence  
et la variable *agetest* change de valeur



```
file:///c:/users/hp/documents/visual studio 2015/Projects/
Je suis le constructeur avec arguments
Nom : El Hani
Prénom : Karima
Age : 31
Après la modification :
Nom : El Hani
Prénom : Karima
Age : 12
agetest=12
```

## ▼ Passage par référence avec le mot clé out

Le mot clé out entraîne le passage des arguments par référence. La situation est similaire à celle du mot clé ref, sauf que ref nécessite que la variable soit initialisée avant d'être transmise. Pour utiliser un paramètre out, la définition de la méthode et la méthode d'appel doivent utiliser explicitement le mot clé out.

```

class OutExample {
    static void Method(out int i)
    {
        i = 44;
    }
    static void Main()
    {
        int value;
        Method(out value);
        // value est maintenant = 44
    }
}

```

#### ▼ Passage de paramètres à une méthode

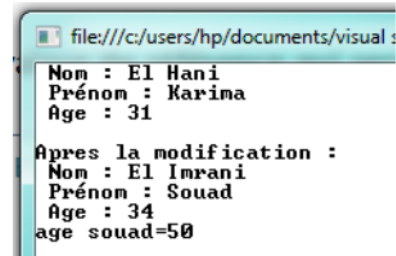
Comme en Java, dans ce cas c'est la valeur de la référence qui sera copiée dans le paramètre formel de la méthode

```

public void InitialiseEtudiant(Etudiant etd)
{
    this.nom = etd.nom;
    this.prenom = etd.prenom;
    this.age = etd.age;

    etd.age = 50;
}
//Méthode de test
public static void Main(string[] args)
{
    Etudiant et1 = new Etudiant("El Hani", "Karima", 31);
    Etudiant et2 = new Etudiant("El Imrani", "Souad", 34);
    et1.afficheEtudiant();
    et1.InitialiseEtudiant(et2);
    Console.WriteLine("\nAprès la modification :");
    et1.afficheEtudiant();
    Console.WriteLine("age souad=" + et2.age);
    Console.Read();
}

```



```

file:///c:/users/hp/documents/visual s
Nom : El Hani
Prénom : Karima
Age : 31

Après la modification :
Nom : El Imrani
Prénom : Souad
Age : 34
age souad=50

```

#### ▼ Méthodes de lecture et d'écriture des attributs privés

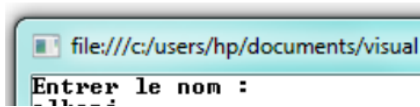
### Méthodes de lecture et d'écriture des attributs privés

Il existe deux méthodes :

- En utilisant les accesseurs et les modificateur (getters/setters) comme en Java
- En utilisant **les propriétés** : Celles-ci permettent de manipuler des attributs privés comme s'ils étaient publics.

```
public class Etudiant
{
    private string nomEtudiant;
    public string nom
    {
        get
        {
            return nomEtudiant;
        }
        set
        {
            nomEtudiant = value;
        }
    }
}

class Test
{
    //Méthode de test
    public static void Main(string[] args)
    {
        Etudiant et1 = new Etudiant();
        Console.WriteLine("Entrer le nom :");
        et1.nom = Console.ReadLine();
        Console.WriteLine("Votre nom est : " + et1.nom);
        Console.ReadLine();
    }
}
```



ou

- public string nom {get; set;}
- public string nom {get; private set;}
- public string nom {private get; set;}

### ▼ heritage

```
public class Chien : Animal
{
    public void Aboier()
    {
        Console.WriteLine(" Haow !");
    }
}
```