



Collections



Introduction aux collections



Définition de la collection en informatique

Une collection est un ensemble d'objets qui sont stockés dans un même conteneur.

Elles sont utilisées pour stocker des données de manière structurée.

Elles possèdent des propriétés et des méthodes qui permettent de les manipuler.





Avantages de l'utilisation des collections en C#

Pourquoi utiliser des collections en C# ?

- ◆ Elles permettent de stocker des données de manière structurée.
- ◆ Elles permettent de manipuler ces données de manière simple et efficace.
- ◆ Elles permettent de stocker des données de types différents.





Les tableaux (Arrays)





Définition et utilisation des tableaux

En C#, un tableau est une structure de données qui permet de stocker un ensemble de valeurs de même type.

Il est déclaré comme suit :

// Explicitement

```
Type[ ] nomTableau = new Type[taille];
```

// Implicitement

```
Type[ ] nomTableau = {valeur1, valeur2, valeur3};
```





Déclaration d'un tableau à plusieurs dimensions

```
// Explicitement  
int[,] tableau = new int[2, 3];  
  
// Implicitement  
int[,] tableau = {{1, 2, 3}, {4, 5, 6}};
```





Boucles `for` et `foreach` pour le parcours

Il est possible de parcourir un tableau avec une boucle for ou foreach.

```
// Boucle for
for (int i = 0; i < tableau.Length; i++)
{
    // Code
}
// Boucle foreach
foreach (Type element in tableau)
{
    // Code
}
```





Propriétés et méthodes des tableaux:





Propriétés:

Les tableaux possèdent des propriétés et des méthodes qui permettent de les manipuler.

Nous pouvons citer les propriétés suivantes :

- ◆ **Length** : retourne la taille du tableau.
- ◆ **Rank** : retourne le nombre de dimensions du tableau.





Length

La propriété Length retourne la taille du tableau.

```
int[] tableau = {1, 2, 3, 4, 5};  
  
Console.WriteLine(tableau.Length); // Affiche 5
```

Cette propriété est utilisée pour parcourir un tableau avec une boucle for.





Rank

La propriété Rank retourne le nombre de dimensions du tableau.

```
int[,] tableau = new int[2, 3];  
  
Console.WriteLine(tableau.Rank); // Affiche 2
```

Pratique pour parcourir un tableau à plusieurs dimensions.





Méthodes

Les tableaux possèdent des méthodes qui permettent de les manipuler.

Parmi elles, nous pouvons citer les méthodes suivantes :

- ◆ `CopyTo()` : copie les éléments d'un tableau
- ◆ `IndexOf()` : retourne l'indice de la première occurrence
- ◆ `Sort()` : trie les éléments du tableau.
- ◆ `BinarySearch()` : recherche un élément dans le tableau.





CopyTo()

La méthode `CopyTo()` permet de copier les éléments d'un tableau dans un autre tableau.

```
int[] tableau1 = {1, 2, 3, 4, 5};  
int[] tableau2 = new int[5];  
  
tableau1.CopyTo(tableau2, 0);  
  
Console.WriteLine(tableau2[0]); // Affiche 1
```





IndexOf()

La méthode `IndexOf()` permet de retourner l'indice de la première occurrence d'un élément dans le tableau.

```
int[] tableau = {1, 2, 3, 4, 5};  
  
Console.WriteLine(tableau.IndexOf(3)); // Affiche 2
```





Sort()

La méthode `Sort()` permet de trier les éléments du tableau.

```
int[] tableau = {5, 4, 3, 2, 1};  
  
tableau.Sort();  
  
Console.WriteLine(tableau[0]); // Affiche 1
```





BinarySearch()

La méthode `BinarySearch()` permet de rechercher un élément dans le tableau.

```
int[] tableau = {1, 2, 3, 4, 5};  
  
Console.WriteLine(tableau.BinarySearch(3)); // Affiche 2
```





Contains()

La méthode Contains() permet de vérifier si un élément est présent dans le tableau.

```
int[] tableau = {1, 2, 3, 4, 5};  
  
Console.WriteLine(tableau.Contains(3)); // Affiche true
```

Pratique pour les lookup tables.





Les listes (Lists)





Définition et utilisation des listes

Les listes sont des structures de données qui permettent de stocker un ensemble de valeurs de même type.

Elles sont déclarées comme suit :

```
// Declaration  
List<Type> nomListe = new List<Type>();  
  
// Declaration et initialisation  
List<Type> nomListe = new List<Type> {valeur1, valeur2, valeur3};
```





Boucles for et foreach pour parcourir les listes

Il est possible de parcourir une liste avec une boucle for ou foreach.

```
// Boucle for
for (int i = 0; i < liste.Count; i++)
{
    // Code
}
// Boucle foreach
foreach (Type element in liste)
{
    // Code
}
```





Propriétés et méthodes des listes:





Propriétés

Les listes possèdent des propriétés et des méthodes qui permettent de les manipuler.

Nous pouvons citer les propriétés suivantes :

- ◆ **Count** : retourne le nombre d'éléments dans la liste.
- ◆ **Capacity** : retourne la capacité de la liste.





Count

La propriété Count retourne le nombre d'éléments dans la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
Console.WriteLine(liste.Count); // Affiche 5
```





Capacity

La propriété Capacity retourne la capacité de la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
Console.WriteLine(liste.Capacity); // Affiche 5
```





Méthodes

Les listes possèdent des méthodes qui permettent de les manipuler.

Parmi elles, nous pouvons citer les méthodes suivantes :

- ◆ `Add()` : ajoute un élément à la fin de la liste.
- ◆ `Insert()` : insère un élément à une position donnée.
- ◆ `Remove()` : supprime la première occurrence d'un élément dans la liste..
- ◆ `Contains()` : retourne vrai si l'élément est présent dans la liste.





Add()

La méthode `Add()` permet d'ajouter un élément à la fin de la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
liste.Add(6);  
  
Console.WriteLine(liste[5]); // Affiche 6
```





Insert()

La méthode `Insert()` permet d'insérer un élément à une position donnée.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
  
liste.Insert(2, 6);  
  
Console.WriteLine(liste[2]); // Affiche 6
```





Remove()

La méthode Remove() permet de supprimer la première occurrence d'un élément dans la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
  
liste.Remove(3);  
  
Console.WriteLine(liste[2]); // Affiche 4
```





RemoveAt()

La méthode `RemoveAt()` permet de supprimer un élément à une position donnée.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
  
liste.RemoveAt(2);  
  
Console.WriteLine(liste[2]); // Affiche 4
```





Clear()

La méthode `Clear()` permet de vider la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
  
liste.Clear();  
  
Console.WriteLine(liste.Count); // Affiche 0
```





Contains()

La méthode `Contains()` permet de vérifier si un élément est présent dans la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
Console.WriteLine(liste.Contains(3)); // Affiche true
```





IndexOf()

La méthode `IndexOf()` permet de retourner l'index de la première occurrence d'un élément dans la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5};  
Console.WriteLine(liste.IndexOf(3)); // Affiche 2
```





LastIndexof()

La méthode `LastIndexof()` permet de retourner l'index de la dernière occurrence d'un élément dans la liste.

```
List<int> liste = new List<int> {1, 2, 3, 4, 5, 3};  
  
Console.WriteLine(liste.LastIndexof(3)); // Affiche 5
```





Sort()

La méthode `Sort()` permet de trier la liste.

```
List<int> liste = new List<int> {5, 2, 3, 4, 1};  
  
liste.Sort();  
  
Console.WriteLine(liste[0]); // Affiche 1
```





Les dictionnaires (Dictionaries)





Définition et utilisation des dictionnaires

Les dictionnaires sont des collections d'éléments qui sont indexés par une clé. Chaque élément est associé à une clé unique.

Les dictionnaires sont très utiles pour stocker des données de type clé/valeur.

Ils permettent de rechercher un élément en fonction de sa clé.





Déclaration et initialisation

```
Dictionary<string, string> dico = new Dictionary<string, string>();  
  
Dictionary<string, string> dico = new Dictionary<string, string>  
{  
    {"cle1", "valeur1"},  
    {"cle2", "valeur2"},  
    {"cle3", "valeur3"}  
};
```





Syntaxe avec `var`

Avec la syntaxe `var`, la déclaration et l'initialisation des dictionnaires sont plus simples.

`var` permet de déclarer une variable sans préciser son type.

```
var dico = new Dictionary<string, string>();  
  
var dico = new Dictionary<string, string>  
{  
    {"cle1", "valeur1"},  
    {"cle2", "valeur2"},  
    {"cle3", "valeur3"}  
};
```





Accès aux éléments d'un dictionnaire

Pour accéder à un élément d'un dictionnaire, il suffit de spécifier sa clé entre crochets, la clef fonctionne comme un index.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

Console.WriteLine(dico["cle1"]); // Affiche valeur1
```





Boucles `foreach` pour parcourir les dictionnaires

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

foreach (KeyValuePair<string, string> element in dico)
{
    Console.WriteLine(element.Key + " : " + element.Value);
}
```





foreach avec var

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

foreach (var element in dico)
{
    Console.WriteLine(element.Key + " : " + element.Value);
}
```





Ajout, mise à jour et suppression d'éléments dans les dictionnaires





Ajout d'éléments dans les dictionnaires

L'ajout d'éléments dans les dictionnaires se fait avec la méthode `Add()`.

Cette méthode prend en paramètre la clé et la valeur.

```
Dictionary<string, string> dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

dico.Add("cle4", "valeur4");
```





Mise à jour d'éléments dans les dictionnaires

La mise à jour d'éléments dans les dictionnaires se fait en spécifiant la clé entre crochets et en lui affectant une nouvelle valeur.

```
Dictionary<string, string> dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

dico["cle1"] = "valeur4";
```





Suppression d'éléments

La suppression d'éléments dans les dictionnaires se fait avec la méthode `Remove()`.

```
Dictionary<string, string> dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

dico.Remove("cle1");
```





Propriétés et méthodes des dictionnaires:



Propriétés

Les dictionnaires possèdent des propriétés qui permettent d'obtenir des informations sur le dictionnaire.

- ◆ `Count` qui retourne le nombre d'éléments du dictionnaire.
- ◆ `Keys` qui retourne une collection des clés du dictionnaire.
- ◆ `Values` qui retourne une collection des valeurs du dictionnaire.





Count

La propriété `Count` retourne le nombre d'éléments du dictionnaire.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

Console.WriteLine(dico.Count); // Affiche 3
```





Keys

La propriété `Keys` retourne une collection des clés du dictionnaire.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

foreach (string cle in dico.Keys)
{
    Console.WriteLine(cle); // Affiche cle1, cle2, cle3
}
```





Values

La propriété `Values` retourne une collection des valeurs du dictionnaire.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
    {"cle3", "valeur3"}
};

foreach (string valeur in dico.Values)
{
    Console.WriteLine(valeur); // Affiche valeur1, valeur2, valeur3
}
```





Parcourir les clés et les valeurs

Il est possible de parcourir les clés et les valeurs d'un dictionnaire en même temps avec la méthode `foreach` et la méthode `KeyValuePair`.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
};

foreach (KeyValuePair<string, string> element in dico)
{
    Console.WriteLine(element.Key + " : " + element.Value);
}
```





Méthodes:

Les dictionnaires possèdent des méthodes qui permettent de manipuler les éléments du dictionnaire.

- ◆ `Add()` qui ajoute un élément au dictionnaire.
- ◆ `Remove()` qui supprime un élément du dictionnaire.
- ◆ `ContainsKey()` qui indique si le dictionnaire contient une clé spécifiée.
- ◆ `ContainsValue()` qui indique si le dictionnaire contient une valeur spécifiée.





Add()

La méthode `Add()` ajoute un élément au dictionnaire.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
};

dico.Add("cle3", "valeur3");
```





Remove()

La méthode `Remove()` supprime un élément du dictionnaire.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
};

dico.Remove("cle1");
```





Clear()

La méthode `Clear()` supprime tous les éléments du dictionnaire.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
};

dico.Clear();
```





ContainsKey()

La méthode `ContainsKey()` indique si le dictionnaire contient une clé spécifiée.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
};

Console.WriteLine(dico.ContainsKey("cle1")); // Affiche True
Console.WriteLine(dico.ContainsKey("cle3")); // Affiche False
```





ContainsValue()

La méthode `ContainsValue()` indique si le dictionnaire contient une valeur spécifiée.

```
var dico = new Dictionary<string, string>
{
    {"cle1", "valeur1"},
    {"cle2", "valeur2"},
};

Console.WriteLine(dico.ContainsKey("valeur1")); // Affiche True
Console.WriteLine(dico.ContainsKey("valeur3")); // Affiche False
```





TryGetValue()

La méthode `TryGetValue()` retourne la valeur associée à la clé spécifiée.

```
string valeur;
if (dico.TryGetValue("cle1", out valeur))
{
    Console.WriteLine(valeur); // Affiche valeur1
}
else
{
    Console.WriteLine("Clé non trouvée");
}
```





Les ensembles (Sets)





Définition et utilisation des ensembles

Un ensemble est une collection d'objets qui ne contient pas de doublons. Il n'y a pas d'ordre dans un ensemble.

```
var set = new HashSet<int> { 1, 2, 3, 4, 5 };
set.Add(1); // Ne fait rien, l'ensemble ne contient pas de doublon
set.Add(6); // Ajoute 6 à l'ensemble

foreach (int i in set)
{
    Console.WriteLine(i);
}
```





Déclaration et initialisation des ensembles

Pour déclarer un ensemble, on utilise la classe `HashSet<T>`.

```
HashSet<T> ensemble = new HashSet<T>();
```





Boucles foreach

Pour parcourir les éléments d'un ensemble, on utilise la boucle `foreach`.

```
foreach (T element in ensemble)
{
    // Code
}
```





Ajout d'éléments dans les ensembles

Pour ajouter un élément dans un ensemble, on utilise la méthode `Add()`.

```
ensemble.Add(element);
```





Suppression d'éléments dans les ensembles

Pour supprimer un élément dans un ensemble, on utilise la méthode `Remove()`.

```
ensemble.Remove(element);
```





Mise à jour d'éléments dans les ensembles

Les HashSet ne permettent pas de mettre à jour un élément.

Pour mettre à jour un élément dans un ensemble, il faut supprimer l'élément puis l'ajouter.

```
ensemble.Remove(element);  
ensemble.Add(element);
```





Propriétés et méthodes des ensembles:



Propriétés Count

La propriété `Count` retourne le nombre d'éléments dans l'ensemble.

```
var ensemble = new HashSet<int> { 1, 2, 3 };  
  
Console.WriteLine(ensemble.Count); // Affiche 3
```





Méthodes:

La classe `HashSet<T>` contient les méthodes suivantes:

- ◆ `Add()`
- ◆ `Clear()`
- ◆ `Contains()`
- ◆ `Remove()`





Clear()

La méthode `Clear()` supprime tous les éléments de l'ensemble.

```
var ensemble = new HashSet<int> { 1, 2, 3 };  
  
ensemble.Clear();
```





Contains()

La méthode `Contains()` indique si l'ensemble contient un élément spécifié.

```
var ensemble = new HashSet<int> { 1, 2, 3 };

// Affiche True
Console.WriteLine(ensemble.Contains(1));
// Affiche False
Console.WriteLine(ensemble.Contains(4));
```

