Fonction (suite)

- Valeur par défaut
- Passage de paramètres par référence
- Surcharge de fonction
- Paramètre de sortie

Valeurs par défaut

- C# permet d'assigner des valeurs par défaut aux paramètres d'une méthode.
- Cela permet de simplifier l'appel de la méthode sans fournir toutes les valeurs.

```
void AfficherMessage(string message, string prefix = "Info") {
    Console.WriteLine($"{prefix}: {message}");
}

AfficherMessage("Ceci est un message."); // Utilise "Info" par défaut
AfficherMessage("Ceci est un avertissement.", "Avertissement");
```

Passage de paramètres par référence en C#

- En C#, les paramètres sont, par défaut, passés par **valeur**, ce qui signifie que la méthode reçoit une **copie** de la variable, sans accès direct à la variable d'origine.
- En utilisant le passage par **référence** (`ref`), la méthode appelée reçoit l'**adresse mémoire** de la variable d'origine. Cela permet de **modifier directement la variable en mémoire**.

Exemple de passage par référence

```
void DoubleValeur(ref int nombre) {
    nombre *= 2;
}
int x = 5;
DoubleValeur(ref x);
Console.WriteLine(x); // Résultat : 10
```

Différence entre copie par valeur et passage par référence

1. **Passage par valeur** :
    - Une **copie** de la valeur d'origine est placée dans un espace mémoire distinct. Les modifications effectuées dans la méthode **ne modifient pas la variable d'origine**.
2. **Passage par référence** (`ref`) :
    - Au lieu de copier la valeur, **l’adresse mémoire** de la variable est transmise.
    - La méthode appelée a donc accès direct à la même adresse en mémoire que la variable originale, ce qui lui permet de **modifier la valeur originale directement**.
    - Ce passage est particulièrement utile lorsqu'on travaille avec des objets volumineux dont la copie prendrait beaucoup de ressource.

Surcharge de fonction

- La surcharge de fonction permet d'avoir plusieurs méthodes avec le même nom mais des signatures différentes.
- Cela peut se faire en variant le type ou le nombre de paramètres.

```
void Afficher(int a) {
    Console.WriteLine($"Nombre entier : {a}");
}

void Afficher(double b) {
    Console.WriteLine($"Nombre à virgule : {b}");
}

void Afficher(string message, int a) {
    Console.WriteLine($"{message} : {a}");
}

Afficher(5);
Afficher(3.14);
Afficher("Nombre entier", 10);
```

Paramètres de sortie

- Les **paramètres de sortie** (`out`) permettent de renvoyer plusieurs valeurs depuis une méthode.
- Les variables de sortie doivent être initialisés dans l'invocation de la méthode pour pouvoir être utilisés.

```
void Calculer(int a, int b, out int somme, out int produit) {
    somme = a + b;
    produit = a * b;
}

Calculer(3, 4, out int resultatSomme, out int resultatProduit);
Console.WriteLine($"Somme: {resultatSomme}, Produit: {resultatProduit}");
```

Notez que les paramètres de sortie sont très peu utilisé. À l'exception de quelque cas très niché, ils sont, selon moi, une béquille en attendant d'utiliser des objets.

Conversion avec `TryParse`

Bien que je vous recommande d'utiliser le moins possible les paramètres de sortie, il est utile de connaitre le principe, puisque ceux-ci sont utilisé par certaine méthode native en C#.

Par exemple, la méthode `TryParse` permet de faire une convertion tout en vérifiant si la conversion est valide, évitant ainsi les exceptions. En cas de succès, la valeur convertie est assignée au paramètre de sortie `out`.

```
string entree = "123";
if (int.TryParse(entree, out int nombre)) {
    Console.WriteLine($"Conversion réussie : {nombre}"); // Affiche : Conversion réussie : 123
}
else {
    Console.WriteLine("La conversion a échoué.");
}
```

Les méthodes `double.TryParse, DateTime.TryParse, Array.TryFindIndex, etc.` utilisent toutes le même principes