

# Fonctionnalités syntaxiques

## Inférence de type

C# supporte l'inférence de type pour les variables locales avec le mot clé `var`, exemple :

```
var hello = "Hello world";
```

A utiliser uniquement lorsque le type est évident (constructor ou littéral).

## Interpolation de chaine

C# supporte l'interpolation de chaine avec `$`, exemple :

```
var firstName = "John";  
var lastName = "Shepard"  
var fullName = $"{firstName} {lastName}";  
Console.WriteLine(fullName); // Prints John Shepard
```

## Opérateurs `?` et `??`

### Nullabilité

Un type suivi de `?` signifie que le type est nullable. En cas d'assignation de `null` à un type qui n'est pas déclaré comme nullable, le compilateur produit un warning. Exemple :

```
int nonNullableNumber= 42;  
nonNullableNumber = null; // warning de compilation  
  
int? nullableNumber = 42;  
nullableNumber = null; // pas de warning de compilation
```

# Ternaires

```
decimal ticketPrice = age >= 18 ? 10 : 6;
```

Equivalent :

```
decimal ticketPrice;  
if(age >= 18){  
    ticketPrice = 10;  
} else {  
    ticketPrice = 6  
}
```

## Accès conditionné par `null`

Permet de conditionner l'accès à un champs ou l'appel d'une méthode par la non présence d'une valeur `null` sur la référence sur laquelle on effectue l'appel. Exemple :

```
User? user = GetUser();  
string? name = user?.Name;
```

Equivalent :

```
User? user = GetUser();  
string? name;  
  
if(user is not null){  
    name = User.Name;  
}
```

## `null`-coalescing

Valeur de repli en cas de valeur `null` :

```
User user = GetUser() ?? new User("John Shepard");
```

Equivalent :

```
User? temp = GetUser();

User user;
if(temp is not null){
    user = temp;
} else {
    user = new User("John Shepard");
}
```

## Méthodes "Expression Body"

```
public class Point2D {
    private int x;
    private int y;

    public Point2D(int x, int y){
        this.x = x;
        this.y = y;
    }

    public Point Move(int distanceX, int distanceY) => new Point(x + distanceX, y + distanceY);
}
```

## Intialisation

## Propriétés d'objets

```
var employee = new Employee{ FirstName = "John", LastName = "Shepard"};
```

Equivalent :

```
var employee = new Employee();
employee.FirstName = "John";
employee.LastName = "Shepard";
```

# Collections

## Listes / Tableaux

```
var list = new List<string> { "Item1", "Item2", "Item3" }
```

## Dictionnaires

```
var map = new Dictionary<int,string> {  
    [42] = "Item1",  
    [41] = "Item2",  
    [123] = "Item3"  
}
```

# Méthodes d'extensions

Les méthodes d'extensions permettent de rajouter des méthodes aux types qu'on ne contrôle pas (ex : classes de la Base Classes Library (BCL) ou des bibliothèques qu'on utilise). Cela requiert de créer une classe `static`, et les méthodes d'extension doivent prendre en premier paramètre un argument du type étendu, avec le mot clé `this`. Exemple :

```
public static class Extensions {  
    public static void Multiply(this string stringToMultiply, int times){  
        return String.Concat(Enumerable.Repeat(stringToMultiply, times));  
    }  
}
```

# Surcharge d'opérateurs

En C#, les opérateurs du langage (+, -, \*, etc...) peuvent être redéfinis :

```
public static Fraction operator +(Fraction a, Fraction b)  
    => new Fraction(a.num * b.den + b.num * a.den, a.den * b.den);
```

---

Revision #1

Created 21 November 2024 14:55:09 by Nicolas

Updated 21 November 2024 14:56:08 by Nicolas