



Presentation du Framework





Historique

Le C# a été créé par Anders Hejlsberg, un des créateurs du langage Turbo Pascal, et est sorti en 2000. Il est basé sur le C++ et le Java.

Le langage a été développé par Microsoft et est utilisé dans le cadre de la plateforme .NET.

Le langage C# est venu du fait que Microsoft voulait un langage qui soit plus facile à utiliser que le C++ et qui soit plus rapide que le Java.





Quelle possibilité offre le C ?

Le C permet de créer :

- ◆ Des applications bureau
- ◆ Des sites internet
- ◆ Des applications mobiles
- ◆ Des jeux vidéos





Application bureau

Il existe plusieurs librairies pour créer des applications bureau :

- ◆ WPF
- ◆ WinForm





Site Internet

Grace a ASP.NET Core, il est possible de creer des sites internet.

Ces sites internet peuvent contenir des pages statiques ou dynamiques.

Ils peuvent communiquer avec une base de données.





Application Mobile

Il est possible de créer des applications mobiles avec Xamarin .

Il est possible de créer des applications mobiles pour Android, IOS et Windows Phone.





Jeux Videos

Il est possible de créer des jeux vidéos avec Unity.

Les plateformes supportées sont Android, IOS, Windows, Linux, Mac OS, WebGL, PS4, Xbox One, Nintendo Switch.





Fonctionnement du C#





Garbage Collector

Le C# utilise un Garbage Collector pour gerer la memoire.

Le Garbage Collector est un programme qui gere la memoire de l'ordinateur a la place de l'utilisateur.

Il permet de liberer de la memoire quand elle n'est plus utilisee.

Cela permet de ne pas avoir a se soucier de la memoire.





Classes et methode

Le C# est un langage oriente objet.

Il se base sur le concept de classe et de methode.

Les fonctions n'existent pas en C#.

Tout est fait a l'aide de classes et de methodes.





Compilation

Le C# est un langage compile et interprete.

Le C# est compile en IL (Intermediate Language) qui est interprete par le CLR (Common Language Runtime).

Ainsi, le C# est un langage compile (MSIL) et interprete (CLR).





Architecture





Assemblies

Le C# est composé d'assemblies, ce sont des sortes de bibliothèques.

Une assembly est un ensemble de classes sous forme de fichier .dll ou .exe.

Les Assemblies sont des unités logiques de distribution et de déploiement dans .NET.

Les assemblies sont utilisés pour la sécurité, la gestion de version et la distribution.





Package

Un package est un ensemble d'assemblies.

Un package C# est un ensemble de bibliothèques et de fichiers de code partagé qui peuvent être utilisés par plusieurs projets

Les packages sont généralement publiés sur un dépôt en ligne tel que NuGet et peuvent être installés dans un projet en quelques clics.





Projet

Un projet C# est un ensemble de fichiers de code, de ressources et de configuration qui forment une application ou une bibliothèque.

Chaque projet C# peut être compilé pour générer un fichier exécutable ou une bibliothèque partagée.





Solutions

Une solution C# est un conteneur qui peut inclure plusieurs projets.

Les solutions permettent aux développeurs de travailler sur plusieurs projets en même temps dans un environnement unique et d'organiser les projets qui font partie d'un système plus large.

En général, une solution C# est utilisée pour les projets de grande envergure qui nécessitent la collaboration de plusieurs développeurs sur plusieurs projets différents





Namespace

Les namespaces en C# sont des espaces de noms.

Ils permettent d'organiser le code et d'éviter les conflits de noms entre les différentes parties d'un projet ou de différents projets.

Les développeurs peuvent utiliser des namespaces pour regrouper des classes, des interfaces et d'autres types de code en groupes cohérents





Bases de la programation





Variable

Une variable est un espace memoire qui peut contenir une valeur.

Une variable est declaree avec un nom et un type.

Il est possible de declarer une variable sans lui donner de valeur.





Syntaxe

```
// Declaration  
type nom_variable;
```

```
// Declaration et initialisation  
type nom_variable = 5;
```

```
// Declaration multiple  
type nom_variable, nom_variable2;
```

```
// Declaration multiple et initialisation  
type nom_variable = 2, nom_variable2 = 5;
```





Type

Le C# est un langage typé.

Il est donc nécessaire de déclarer le type de la variable.

Il existe plusieurs types et nous pouvons en créer de nouveaux.

Les types sont importants pour toutes les variables, retour de fonction, paramètres, etc.





Type de base

Nom	Description	Exemple
int	Entier	5, 10,-100
float	Nombre a virgule	5.5 , -100, 12.30545
double	Nombre a virgule double precision	5.5 , -100, 12.30545
char	Caractere	'a' , '+', 'F'
string	Chaine de caractere	"Hello" , "World\n"
bool	Booleen	true , false





Nullable type

Le C# permet de créer des types nullable.

Cela permet de pouvoir mettre une valeur `null` dans une variable.

Il suffit de rajouter un `?` après le type.

```
int? a = null;  
double? b = null;  
bool? c = null;
```





Operateurs

Les operateurs sont des symboles qui permettent de faire des operations.

Il 3 types d'operateurs :

- ◆ Arithmetiques
- ◆ Comparaison
- ◆ Logique





Arithmetiques

Ces operateurs permettent de faire des operations arithmetiques.

Nom	Description	Exemple
+	Addition	$5 + 5$
-	Soustraction	$5 - 5$
*	Multiplication	$5 * 5$
/	Division	$5 / 5$
%	Modulo	$5 \% 5$





Comparaison

Ce type d'opérateur permet de comparer des valeurs.

Nom	Description	Exemple
<code>==</code>	Egal	<code>5 == 5</code>
<code>!=</code>	Different	<code>5 != 5</code>
<code>></code>	Supérieur	<code>5 > 5</code>
<code><</code>	Inferieur	<code>5 < 5</code>
<code>>=</code>	Supérieur ou égal	<code>5 >= 5</code>
<code><=</code>	Inferieur ou égal	<code>5 <= 5</code>





Logique

Ce type d'opérateur permet de faire des opérations logiques.

Nom	Description	Exemple
&&	ET	true && true
	OU	true false
!	NON	!false





Condition





Conditions SI , SI NON, SI NON SI

Pour faire des conditions, on utilise les mots clés `if` , `else if` et `else` .

```
if (condition){  
    //Code  
}  
else if (condition){  
    //Code  
}  
else{  
    //Code  
}
```





Switch

Afin de faire des conditions plus complexes, il y a le mot clé `switch`.

```
switch (variable){  
    case valeur1:  
        //Code  
        break;  
    case valeur2:  
        //Code  
        break;  
    default:  
        //Code  
        break;  
}
```





Ternaire

Le C# permet de faire des conditions ternaires.

```
variable = condition ? valeur1 : valeur2;
```

Cela revient à faire :

```
if (condition){  
    variable = valeur1;  
}  
else{  
    variable = valeur2;  
}
```





Boucles





For

Le for est une boucle qui permet de faire une action un nombre de fois défini.

```
for (initialisation; condition; incrementation){  
    //Code  
}
```

Exemple :

```
for (int i = 0; i < 10; i++){  
    //Code  
}
```





While

Le while est une boucle qui permet de faire une action tant qu'une condition est vrai.

```
while (condition){  
    //Code  
}
```





Do-while

Le do-while est une boucle qui permet de faire une action tant qu'une condition est vrai.

La difference avec le while est que le code est executer au moins une fois.

```
do{  
    //Code  
}while(condition);
```





Tableau

Les tableaux sont des variables qui permettent de stocker plusieurs valeurs.

Pour créer un tableau, il faut utiliser le mot clé `new`.

```
type[] tableau = new type[Taille];
```

Exemple :

```
int[] tableau = new int[10];
```





Parcourir un tableau

Pour parcourir un tableau, il existe plusieurs méthodes:

- ◆ Méthode 1 : `foreach`, cette méthode est la plus simple, elle permet de parcourir le tableau sans avoir à se soucier de l'index.
- ◆ Méthode 2 : `for`, cette méthode est plus complexe, elle permet de parcourir le tableau mais nécessite de connaître l'index.





Methode 1 : foreach

Pour parcourir un tableau avec la methode `foreach`, il suffit de faire :

```
foreach (type variable in tableau){  
    //Code  
}
```

La variable prendra la valeur de chaque element du tableau.

Changez la valeur de la variable ne changera pas la valeur de l'element du tableau.





Methode 2 : `for` ou `while`

Pour parcourir un tableau avec la méthode `for` ou `while`, il suffit de faire :

```
for (int i = 0; i < tableau.Length; i++) {
    //Code
}
//ou
int i = 0;
while (i < tableau.Length){
    //Code
    i++;
}
```





Quelques méthodes utiles

Les tableaux ont quelques méthodes / propriétés utiles :

- ◆ `tableau.Length` : Donne la taille du tableau.
- ◆ `tableau.Contains(valeur)` : permet de savoir si un élément est dans le tableau.
- ◆ `tableau.IndexOf(valeur)` : permet de connaître l'index d'un élément dans le tableau.





Fonction





Qu'est ce qu'une fonction

Une fonction est une partie de code qui peut etre appeler plusieurs fois.

Elles permettent de factoriser du code, et de le rendre plus lisible.

Une fonction peut prendre des parametres, et retourner une valeur.





Syntaxe

Une fonction est déclarée comme suit :

```
type NomFonction() {  
    //Code  
}
```

Elle est appellée comme suit :

```
NomFonction();
```





Parametres

Une fonction peut prendre des parametres.

Un parametre est une variable qui est donne a la fonction.

```
type NomFonction(type parametre){  
    //Code  
}  
  
// Nous pouvons avoir plusieurs parametres.  
  
type NomFonction(type parametre1, type parametre2){  
    //Code  
}
```





Parametre par defaut

Une fonction peut avoir des parametres par defaut.

Un parametre par defaut est une valeur qui est donne a la fonction si le parametre n'est pas donne.

```
type NomFonction(type parametre1, type parametre2 = valeur){  
    //Code  
}
```





Appeler une fonction avec des paramètres

Une fonction qui a besoin de 2 paramètres est appeler comme suit :

```
NomFonction(valeur1, valeur2);
```

Une fonction qui a un parametre par defaut est appeler comme suit :

```
NomFonction(valeur1);    // valeur2 prendra la valeur  
                        // par defaut a l'interieur de la fonction.
```





Retourner une valeur

Une fonction peut retourner une valeur.

```
type NomFonction() {  
    //Code  
    return valeur;  
}
```

Pour recuperer la valeur retourner par la fonction, il suffit de la placer dans une variable.

```
type valeur = NomFonction();
```





Overloading

Une fonction peut etre declaree plusieurs fois avec le meme nom.

Cela permet de faire plusieurs fonctions qui ont le meme nom mais qui ont des parametres differents, c'est l'overloading.

```
type NomFonction(type parametre1){  
    //Code  
}  
  
type NomFonction(type parametre1, type parametre2){  
    //Code  
}
```





Lambda

Une fonction lambda est une fonction qui n'a pas de nom.

Elle est déclarée comme suit :

```
(paramètres) => expression de retour
```

Les lambda sont souvent utilisées avec les fonctions de tri :

```
int[] tableau = {1, 2, 3, 4, 5};  
tableau = tableau.OrderBy(x => x).ToArray();
```





Exception

Il arrive que des erreurs se produisent lors de l'exécution d'un programme.

Ces erreurs peuvent mettre le programme en panne, ou le faire planter.

Il est donc important de les gérer.





Try catch

Pour gerer une erreur, on utilise le `try catch`.

```
try{  
    //Code qui peut provoquer une erreur  
}catch(Exception e){  
    //Code qui sera execute si une erreur se produit  
}finally{  
    //Code qui sera execute peu importe  
    // si une erreur se produit ou non  
}
```





Définir ses propres exceptions

Il est possible de définir ses propres exceptions comme suit :

```
public class MaPropreException : Exception{  
    public MaPropreException(string message) : base(message){  
    }  
}
```





Containers utiles





List

La classe List est une classe qui permet de stocker des elements.

Elle permet de faire des operations sur les elements, tel que les trier, les inverser, les melanger, etc.

Elle est declaree comme suit :

```
List<type> liste = new List<type>();
```

Son role est de stocker des elements, et de les recuperer.





Dictionnaire

La classe Dictionnaire est une classe qui permet de stocker des éléments.

La particularité de cette classe est qu'elle permet de stocker des éléments avec une clé et une valeur.

Elle est déclarée comme suit :

```
Dictionnaire<typeClé, typeValeur> dictionnaire = new Dictionnaire<typeClé, typeValeur>();
```





HashSet

La classe HashSet est une classe qui permet de stocker des éléments.

La particularité de cette classe est qu'elle permet de stocker des éléments sans doublons.

L'utilisation de HashSet est très optimisée, et permet de stocker beaucoup d'éléments.

Elle est déclarée comme suit :

```
HashSet<type> hashset = new HashSet<type>();
```





Quelques API Utiles





Console

Nom méthode	Prototype	Explication
WriteLine	Console.WriteLine(string message)	Affiche un message dans la console
ReadLine	Console.ReadLine()	Attend que l'utilisateur entre une valeur dans la console
Clear	Console.Clear()	Efface le contenu de la console
ForegroundColor	Console.ForegroundColor = ConsoleColor.Red;	Change la couleur du texte de la console
BackgroundColor	Console.BackgroundColor = ConsoleColor.Red;	Change la couleur de fond de la console





String

Nom methode	Prototype	Explication
Split	<code>string.Split(char[] separator)</code>	Découpe une chaîne de caractères en plusieurs sous chaînes
ToUpper	<code>string.ToUpper()</code>	Met la chaîne de caractères en majuscule
ToLower	<code>string.ToLower()</code>	Met la chaîne de caractères en minuscule
Trim	<code>string.Trim()</code>	Enlève les espaces en début et fin de chaîne
Replace	<code>string.Replace(char oldChar, char newChar)</code>	Remplace un caractère par un autre caractère





Math

Nom methode	Prototype	Explication
Abs	<code>Math.Abs(int nombre)</code>	Retourne la valeur absolue d'un nombre
Max	<code>Math.Max(int nombre1, int nombre2)</code>	Retourne le plus grand nombre
Min	<code>Math.Min(int nombre1, int nombre2)</code>	Retourne le plus petit nombre
Pow	<code>Math.Pow(int nombre, int exposant)</code>	Retourne le nombre à la puissance de l'exposant
Sqrt	<code>Math.Sqrt(int nombre)</code>	Retourne la racine carrée d'un nombre





Random

La classe Random permet de generer des nombres aleatoires.

Elle est declaree comme suit :

```
Random random = new Random();
```

Elle permet de generer des nombres aleatoires entre 1 et 10 comme suit :

```
int nombreAleatoire = random.Next(1, 10);
```





Enums

Les enums sont des types de données qui permettent de stocker des valeurs prédefinies.

Elles sont pratique pour stocker des valeurs qui ne changent pas.

```
enum Couleur{
    ROUGE,    // 0
    VERT,     // 1
    BLEU      // 2
}
```





References

Lorsque l'on place une variable en parametre d'une methode, on ne place pas reellement la variable, mais une copie de la variable.

Cela permet de proteger la variable d'etre modifiee par la methode.

Il est donc impossible de modifier la variable dans la methode.

```
int nombre = 10;  
ModifierNombre(nombre);  
Console.WriteLine(nombre); // 10
```





References

Cependant, il est possible de modifier la variable en utilisant une reference.

Pour cela, il faut utiliser le mot clef `ref`.

```
int nombre = 10;  
ModifierNombre(ref nombre);  
Console.WriteLine(nombre); // 20
```

