

Programmer en C# - Créer un projet Console (.NET)

Présentation

C# est un langage de programmation orienté objet développé par Microsoft au sein de la plateforme .NET. Il permet de créer des applications console, desktop, web ou mobiles. Un projet console est le point de départ pour découvrir la syntaxe de base du langage et la structure d'une application C#.

Outils

- Télécharger et installer Visual Studio (Community Edition suffit).
- S'assurer d'avoir installé la charge de travail Développement Desktop .NET.
- Vérifier l'installation avec :
 - `dotnet --version` dans un terminal ou PowerShell.

Prérequis

- Connaître la base des types de données en C#.
- Comprendre la notion de programme séquentiel (exécution ligne par ligne).

Créer un projet

- Ouvrir Visual Studio depuis le menu démarrer.
- Cliquer sur Créer un nouveau projet.
- Rechercher "Console" puis choisir Application Console (.NET Core) ou Application Console (.NET) selon les versions.
- Cliquer sur Suivant.
- Nom du projet :
 - donner un nom explicite, exemple MonPremierProjetConsole.
- Emplacement :
 - choisir un dossier pour enregistrer.
- Solution :
 - créer une nouvelle solution.
- Cliquer sur Créer.
- Visual Studio génère automatiquement :
 - un fichier Program.cs
 - un fichier .csproj (paramètres du projet)

Affichage en Console

<code>Console.WriteLine(valeur);</code>	Affiche un texte suivi d'un retour à la ligne.
<code>Console.Write(valeur);</code>	Affiche un texte sans retour à la ligne.
<code>Console.Clear();</code>	Efface tout ce qui est affiché dans la console.

- Exemples :
`Console.WriteLine("Bonjour !");`
`Console.Write("Entrez votre prénom : ");`
`Console.Clear(); // Efface l'écran`

Programmer en C# - Créer un projet Console (.NET)

Types de Données

Voici les types les plus courants en C# :

Type de base	Alias .NET	Taille	Exemples de valeurs
int	System.Int32	4 octets	0, -10, 2456
double	System.Double	8 octets	3.14, -75.8
bool	System.Boolean	1 octet	true, false
char	System.Char	2 octets	'A', 'b'
string	System.String	Variable	"Bonjour", "1234"

Déclaration et Affectation de Variables

Déclaration seule :

```
int age;
```

Affectation seule :

```
age = 18;
```

Déclaration + affectation :

```
int age = 18;
```

Exemples supplémentaires :

```
string nom = "Alice";
```

```
bool majeur = true;
```

```
double taille = 1.75;
```

Opérations sur Variables

Catégorie	Opérateurs	Exemple en C#
Addition	+	a + b
Soustraction	-	a - b
Multiplication	*	a * b
Division	/	a / b
Modulo (reste)	%	a % b
Incrémentation	++	a++
Décrémentation	--	a--

Exemples : `int a = 5, b = 3;`

```
Console.WriteLine(a + b); => 8
```

```
Console.WriteLine(a % b); => 2
```

Programmer en C# - Créer un projet Console (.NET)

Lecture clavier (Input utilisateur)

Lecture d'une ligne de texte :

```
string saisie = Console.ReadLine();  
Console.WriteLine("Vous avez saisi : " + saisie);
```

Lecture et conversion en entier :

```
Console.WriteLine("Entrez un nombre : ");  
int nombre = Convert.ToInt32(Console.ReadLine());  
Console.WriteLine("Le double est : " + (nombre * 2));
```

Conditions

Structure if simple :

```
if (age >= 18)  
{  
    Console.WriteLine("Vous êtes majeur.");  
}
```

If - else :

```
if (age >= 18)  
{  
    Console.WriteLine("Majeur");  
}  
else  
{  
    Console.WriteLine("Mineur");  
}
```

If - else if - else :

```
if (age < 12)  
{  
    Console.WriteLine("Enfant");  
}  
else if (age < 18)  
{  
    Console.WriteLine("Adolescent");  
}  
else  
{  
    Console.WriteLine("Adulte");  
}
```

Programmer en C# - Créer un projet Console (.NET)

Switch

```
Console.WriteLine("Entrez une note entre 1 et 5 : ");  
int note = Convert.ToInt32(Console.ReadLine());
```

```
switch (note)  
{  
    case 1:  
        Console.WriteLine("Très mauvais");  
        break;  
    case 5:  
        Console.WriteLine("Excellent !");  
        break;  
    default:  
        Console.WriteLine("Note correcte");  
        break;  
}
```

Boucle for (connue)

```
for (int i = 0; i < 5; i++)  
{  
    Console.WriteLine("i vaut : " + i);  
}
```

Boucle while (tant que)

```
int compteur = 0;  
while (compteur < 3)  
{  
    Console.WriteLine("Compteur : " + compteur);  
    compteur++;  
}
```

Boucle do-while (au moins une fois)

```
int nb;  
do  
{  
    Console.WriteLine("Tapez un nombre supérieur à 10 : ");  
    nb = Convert.ToInt32(Console.ReadLine());  
} while (nb <= 10);
```

Programmer en C# - Créer un projet Console (.NET)

Méthodes (Fonctions) en C#

Une méthode (aussi appelée fonction) est un bloc de code réutilisable qui exécute une série d'instructions.

Elle peut :

- soit retourner une valeur,
- soit ne rien retourner (void).

Pourquoi utiliser des méthodes ?

- Organiser le code en petites parties claires.
- Éviter de répéter du code.
- Rendre le code plus lisible et modifiable.

Structure d'une Méthode

modificateur typeDeRetour NomDeLaMéthode(paramètres)

```
{  
    // Corps de la méthode  
}
```

Élément	Signification
modificateur	Indique la visibilité (ex : public, private)
typeDeRetour	Type de la valeur retournée (ex : int, string, ou void)
NomDeLaMéthode	Nom donné à la méthode (par convention : commencer par une majuscule)
paramètres	Variables passées à la méthode

Exemple de Méthode sans Retour

```
public static void DireBonjour()  
{  
    Console.WriteLine("Bonjour !");  
}
```

Utilisation (appel de méthode) :

```
DireBonjour();
```

Exemple de Méthode avec Retour

```
public static int Additionner(int a, int b)  
{  
    return a + b;  
}
```

Utilisation (appel de méthode) :

```
int resultat = Additionner(5, 3);  
Console.WriteLine("Résultat : " + resultat);
```

Programmer en C# - Créer un projet Console (.NET)

Passage de Paramètres

Passage	Explication	Exemple
Par valeur	Envoie une copie de la valeur.	Additionner(5, 3)
Par référence (ref, out)	Envoie l'adresse mémoire (modif	Avancé (pas nécessaire pour un c

Modificateurs courants des méthodes

Modificateur	Utilité
public	Accessible de partout
private	Accessible seulement dans la même classe
static	Peut être appelée sans créer un objet

Exemple avec plusieurs modificateurs :

```
public static void AfficherBienvenue()  
{  
    Console.WriteLine("Bienvenue !");  
}
```

Tableaux (Arrays) en C#

Un tableau est une structure de données qui permet de stocker plusieurs valeurs du même type dans une seule variable.

→ En C#, les tableaux sont de taille fixe (on doit définir leur taille à la création).

Déclaration d'un Tableau

Syntaxe :

```
type[] nomTableau;
```

Exemple :

```
int[] notes;
```

→ Ici, notes est déclaré comme un tableau d'entiers (int), mais il n'est pas encore créé en mémoire.

Initialisation d'un Tableau

Avec une taille fixe :

```
notes = new int[5];
```

→ Cela crée un tableau de 5 entiers (par défaut, tous les éléments sont à 0).

Déclaration + initialisation directe :

```
int[] notes = new int[5];
```

Programmer en C# - Créer un projet Console (.NET)

Avec des valeurs directement assignées :

```
int[] notes = { 12, 15, 8, 19, 14 };
```

Accéder aux Éléments

Syntaxe :

```
nomTableau[indice]
```

- Les indices commencent à 0 en C#.

Exemple :

```
Console.WriteLine(notes[0]); // Affiche 12
```

```
Console.WriteLine(notes[3]); // Affiche 19
```

Modifier un élément :

```
notes[2] = 10; // Change la 3ème note (indice 2) à 10
```

Parcourir un Tableau

Avec une boucle for classique :

```
for (int i = 0; i < notes.Length; i++)
```

```
{
```

```
    Console.WriteLine("Note " + (i+1) + " : " + notes[i]);
```

```
}
```

→ .Length donne la taille totale du tableau.

Avec une boucle foreach :

```
foreach (int note in notes)
```

```
{
```

```
    Console.WriteLine("Note : " + note);
```

```
}
```

→ Le foreach passe automatiquement sur chaque élément.

Exemple Complet

```
int[] notes = { 12, 15, 8, 19, 14 };
```

```
// Afficher toutes les notes
```

```
for (int i = 0; i < notes.Length; i++)
```

```
{
```

```
    Console.WriteLine("Note numéro " + (i+1) + " : " + notes[i]);
```

```
}
```

```
// Calculer la moyenne
```

```
int somme = 0;
```

```
for (int i = 0; i < notes.Length; i++)
```

```
{
```

```
    somme += notes[i];
```

```
}
```

```
double moyenne = (double)somme / notes.Length;
```

```
Console.WriteLine("Moyenne des notes : " + moyenne);
```

Programmer en C# - Créer un projet Console (.NET)

Résumé sur les Tableaux

Action	Syntaxe	Exemple
Déclarer un tableau	<code>type[] nom;</code>	<code>int[] notes;</code>
Créer un tableau	<code>new type[taille];</code>	<code>new int[5];</code>
Initialiser directement	<code>{ valeur1, valeur2, ... }</code>	<code>{ 12, 15, 8 }</code>
Accéder à un élément	<code>nom[indice]</code>	<code>notes[0]</code>
Parcourir (for)	<code>for (int i = 0; i < notes.Length; i++)</code>	
Parcourir (foreach)	<code>foreach (type elem in tableau)</code>	

Les Collections en C#

→ Une collection est un ensemble d'objets stockés ensemble, mais plus souple qu'un tableau classique.

Différences par rapport aux tableaux :

Tableau	Collection
Taille fixée à la création	Taille dynamique (ajouter/enlever des éléments facilement)
Stocke des types simples	Stocke des objets ou types complexes
Syntaxe simple mais limitée	Plus de fonctionnalités

Le type List<T>

La List est la collection la plus utilisée.

→ C'est comme un tableau mais qui peut grandir ou rétrécir automatiquement.

Syntaxe pour déclarer une liste :

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

Exemples :

```
List<int> notes = new List<int>();
```

```
List<string> prenom = new List<string>();
```


Programmer en C# - Créer un projet Console (.NET)

Ajouter des éléments dans une List

Ajouter un élément :

```
notes.Add(15);
notes.Add(18);
prenoms.Add("Alice");
```

Accéder aux éléments

Accès par l'indice (comme un tableau) :

```
Console.WriteLine(notes[0]); // Affiche 15
Console.WriteLine(prenoms[1]); // Affiche "Alice"
```

Parcourir une List

Avec une boucle for :

```
for (int i = 0; i < notes.Count; i++)
{
    Console.WriteLine("Note " + (i + 1) + " : " + notes[i]);
}
```

Avec une boucle foreach :

```
foreach (int note in notes)
{
    Console.WriteLine("Note : " + note);
}
```

Rappel :

- .Count donne le nombre d'éléments dans une List.

Supprimer des éléments

Supprimer un élément par valeur :

```
notes.Remove(15);
```

→ Supprime la première occurrence de 15.

Supprimer un élément par index :

```
notes.RemoveAt(0); // Supprime l'élément à l'indice 0
```

Vider toute la liste :

```
notes.Clear();
```

Chercher dans une List

Vérifier si un élément existe :

```
bool existe = notes.Contains(18);
```

Trouver l'index d'un élément :

```
int index = notes.IndexOf(18);
```

→ Retourne -1 si l'élément n'existe pas.

Programmer en C# - Créer un projet Console (.NET)

Exemple Complet

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main(string[] args)
    {
        List<string> fruits = new List<string>();

        fruits.Add("Pomme");
        fruits.Add("Banane");
        fruits.Add("Cerise");

        foreach (string fruit in fruits)
        {
            Console.WriteLine(fruit);
        }

        fruits.Remove("Banane");

        Console.WriteLine("Après suppression :");
        foreach (string fruit in fruits)
        {
            Console.WriteLine(fruit);
        }
    }
}
```

Sortie console :

Pomme

Banane

Cerise

Après suppression :

Pomme

Cerise

Programmer en C# - Créer un projet Console (.NET)

Résumé sur List<T>

Action	Syntaxe
Créer une liste	List<int> liste = new List<int>();
Ajouter	liste.Add(valeur);
Accéder	liste[index]
Supprimer par valeur	liste.Remove(valeur);
Supprimer par index	liste.RemoveAt(index);
Vider la liste	liste.Clear();
Taille de la liste	liste.Count
Contient un élément ?	liste.Contains(valeur)

Programmation Orientée Objet (POO) en C#

La Programmation Orientée Objet (POO) est un style de programmation où l'on organise son code en objets.

Chaque objet représente :

- un état (avec ses propriétés ou attributs),
- des comportements (avec ses méthodes).

Exemple concret :

- Objet réel : Voiture
 - Propriétés : couleur, marque, modèle
 - Méthodes : démarrer(), freiner()

En POO, les classes sont des modèles pour créer ces objets.

Concepts Fondamentaux

Classes et Objets

- Classe = Modèle qui définit les attributs et méthodes.
- Objet = Instance créée à partir de la classe.

Exemple en C# :

```
public class Voiture
{
    public string Marque;
    public string Modele;
}
```

Programmer en C# - Créer un projet Console (.NET)

Créer un objet :

```
Voiture v1 = new Voiture();  
v1.Marque = "Renault";  
v1.Modele = "Clio";
```

Propriétés et Méthodes

- Propriétés : décrivent les caractéristiques d'un objet.
- Méthodes : actions que l'objet peut réaliser.

Exemple :

```
public class Voiture  
{  
    public string Marque;  
  
    public void Klaxonner()  
    {  
        Console.WriteLine("Bip Bip !");  
    }  
}
```

Visibilité (Encapsulation)

Modificateur	Signification
public	Accès partout
private	Accès uniquement dans la classe
protected	Accès dans la classe et ses classes enfants

Exemple :

```
public class CompteBancaire  
{  
    private double solde; // Accessible uniquement à l'intérieur  
}
```

Accesseurs : Getters et Setters

Permettent de lire/écrire des propriétés privées de manière contrôlée.

Programmer en C# - Créer un projet Console (.NET)

Exemple classique :

```
public class CompteBancaire
{
    private double solde;

    public double GetSolde()
    {
        return solde;
    }

    public void SetSolde(double valeur)
    {
        solde = valeur;
    }
}
```

Exemple moderne (C# propriété automatique) :

```
public double Solde { get; set; }
```

Concepts Avancés

Constructeurs

Un constructeur est une méthode spéciale appelée automatiquement quand un objet est créé.

Exemple :

```
public class Voiture
{
    public string Marque;

    // Constructeur
    public Voiture(string marque)
    {
        Marque = marque;
    }
}
```

Utilisation :

```
Voiture maVoiture = new Voiture("Peugeot");
```

Héritage

L'héritage permet de créer une nouvelle classe basée sur une classe existante.

- La classe d'origine = classe mère (parent)
- La nouvelle classe = classe fille (enfant)

Programmer en C# - Créer un projet Console (.NET)

Exemple :

```
public class Vehicule
{
    public void Rouler()
    {
        Console.WriteLine("Le véhicule roule");
    }
}
```

```
public class Voiture : Vehicule
{
    public void Klaxonner()
    {
        Console.WriteLine("Bip Bip !");
    }
}
```

Utilisation :

```
Voiture v = new Voiture();
v.Rouler();    // Hérité de Vehicule
v.Klaxonner(); // Spécifique à Voiture
```

Abstraction

Une classe abstraite sert de modèle mais ne peut pas être instanciée directement.

Exemple :

```
public abstract class Animal
{
    public abstract void Crier();
}
```

```
public class Chien : Animal
{
    public override void Crier()
    {
        Console.WriteLine("Ouaf !");
    }
}
```

Polymorphisme

Le polymorphisme permet de traiter plusieurs objets différents de manière uniforme.

Exemple :

```
Animal monAnimal = new Chien();
monAnimal.Crier(); // Affiche "Ouaf !"
Même si monAnimal est un Animal, il appelle le comportement spécifique de Chien.
```

Programmer en C# - Créer un projet Console (.NET)

Interfaces

Une interface est un contrat : elle liste des méthodes qu'une classe devra obligatoirement implémenter.

Exemple :

```
public interface IVehicule
{
    void Demarrer();
}
```

```
public class Moto : IVehicule
{
    public void Demarrer()
    {
        Console.WriteLine("Moto démarre !");
    }
}
```

Utilisation :

```
Moto m = new Moto();
m.Demarrer();
```

Concepts Avancés

Concept	Exemple	Rôle principal
Classe	class Voiture	Modèle pour créer des objets
Objet	new Voiture()	Instance d'une classe
Héritage	class Moto : Vehicule	Réutiliser du code
Abstraction	abstract class Animal	Obliger à compléter le comportement
Polymorphisme	Animal a = new Chien()	Adapter selon le type réel
Interface	interface IVehicule	Définir un ensemble d'obligations

Modèle de bloc try-catch en C#

```
try {
    // Code à exécuter qui pourrait provoquer une exception
} catch (Exception ex) {
    // Traitement de l'exception
}
```
