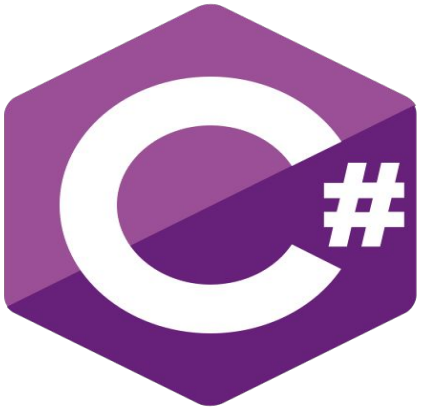
epsi
l'école d'ingénierie informatique

Pro Alterna®



Fondamentaux

Programme de formation Socle Numérique 2ème année

Objectifs du Cours

- Comprendre les concepts de base de la programmation en C#
- Écrire des programmes simples en C#
- Acquérir une pratique concrète grâce à des exercice
- Préparer le terrain pour des concepts plus avancés

Présentation du langage C# et de son environnement de développement

C# (prononcé "C Sharp") est un langage de programmation moderne, orienté objet, développé par Microsoft dans les années 2000.

Il fait partie de la plateforme .NET, qui fournit un ensemble d'outils et de bibliothèques pour développer des applications.

Utilisations courantes :

- Applications de bureau (Windows Forms, WPF).
- Applications web (ASP.NET).
- Jeux vidéo (via Unity).
- Applications mobiles (Xamarin).

Petite récap pour C++ et JAVA

- **Le langage C++** a été créé vers 1980. Il est basé sur le langage C, qui a été inventé au début des années 1970.
- **Le C++** est un langage de programmation orienté objet, générique et multi-paradigme, qui offre une grande performance et un contrôle bas niveau.
- **Le langage Java** a été créé par James Gosling et deux de ses collègues chez Sun Microsystems en 1991. Il est également inspiré par la syntaxe du langage C, mais il est plus simple et plus portable.
- **Le Java** est un langage de programmation orienté objet, multi-paradigme et multiplateforme, qui offre une grande sécurité et une grande compatibilité.

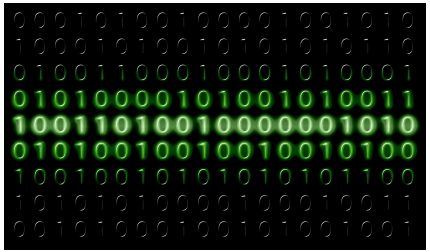
Une application informatique

L’ordinateur exécute des applications informatiques pour effectuer des tâches. Ce sont des logiciels comme :

- Un traitement de texte
- Un navigateur internet
- Un jeu vidéo…..

L’ordinateur ne peut exécuter ces applications informatiques que si elles sont écrites dans le seul langage qu'il comprend, le binaire:

- Techniquement, le binaire est représenté par une suite de 0 et de 1



➔ Il n'est bien sûr pas raisonnablement possible de réaliser une grosse application en binaire, c'est pour ça qu'il existe des langages de programmation qui permettent de simplifier l'écriture d'une application informatique

Créer un programme

le principe de fonctionnement des langages "traditionnels" comme le C et le C++, puis je vous présenterai le fonctionnement du C#. Comme le C# est plus récent, il a été possible d'améliorer son fonctionnement par rapport au C et au C++ comme nous allons le voir

Langages traditionnels : **la compilation**

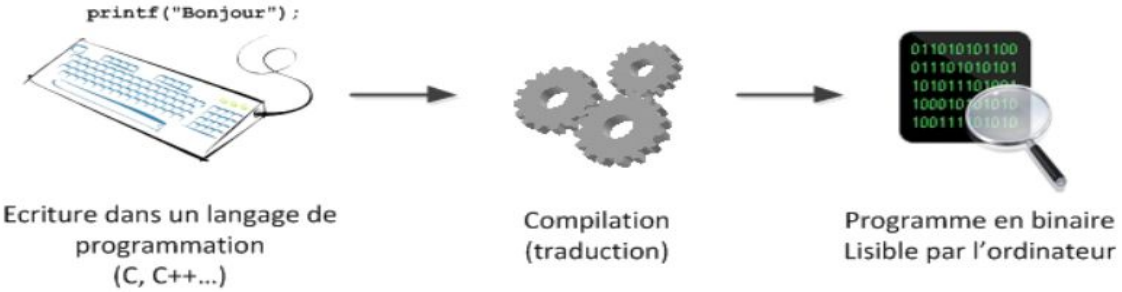
- Avec des langages traditionnels comme le C et le C++, on écrit des instructions simplifiées, lisibles par un humain comme :

Code C :

- printf(“Bonjour”);

Créer un programme

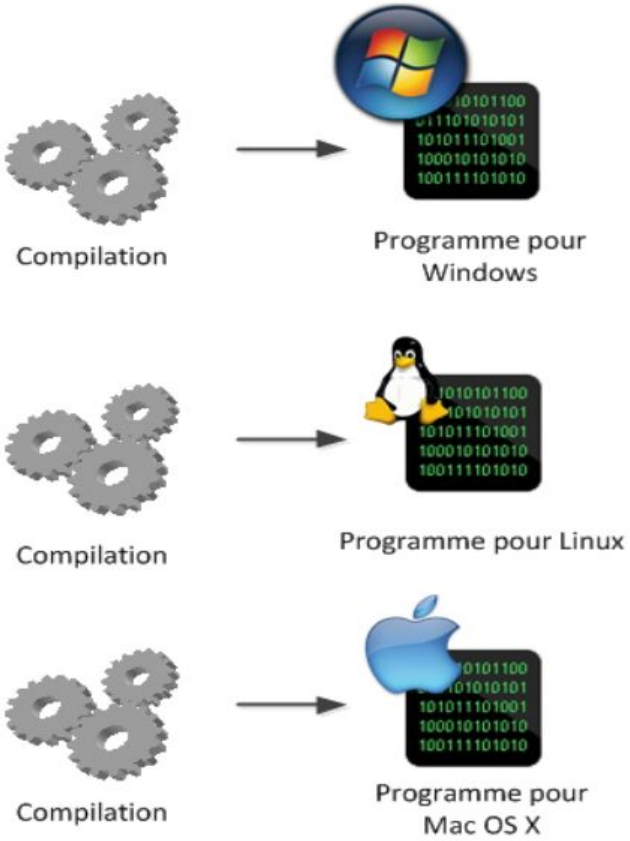
- L'ordinateur ne comprend pas ces instructions. Lui, il veut du binaire, du vrai.
- Pour obtenir du binaire à partir d'un code écrit en C ou C++, on doit effectuer ce qu'on appelle une compilation.
- Le compilateur est un programme qui traduit le code source en binaire exécutable :



Output: Bonjour****

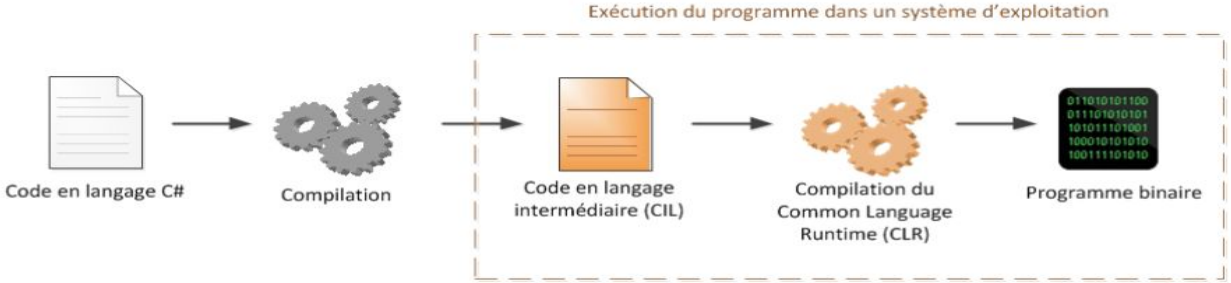
Créer un programme

- Cette méthode est efficace et a fait ses preuves. De nombreuses personnes développent toujours en C et C++ aujourd'hui.
- Néanmoins, ces langages ont aussi un certain nombre de défauts dus à leur ancienneté.Par exemple, un programme compilé (binaire) ne fonctionne que sur la plateforme pour laquelle il a été compilé. Cela veut dire que si vous compilez sous Windows,vous obtenez un programme qui fonctionne sous Windows uniquement (et sur un type de processeur particulier). Impossible de le faire tourner sous Mac OS X ou Linux simplement, à moins de le recompiler sous ces systèmes d'exploitation (et d'effectuer au passage quelques modifications)



Créer un programme

- **Les programmes binaires ont ce défaut :** ils ne fonctionnent que pour un type de machine. Pour les développeurs qui écrivent le code, c'est assez fatiguant à gérer.
- **Langages récents :** le code managé Les langages récents, comme le C# et le Java résolvent ce problème de compatibilité tout en ajoutant de nombreuses fonctionnalités appréciables au langage, ce qui permet de réaliser des programmes beaucoup plus efficacement.
- **La compilation** en C# ne donne pas un programme binaire, contrairement au C et au C++. Le code C# est en fait transformé dans un langage intermédiaire (appelé CIL ou MSIL) que l'on peut ensuite distribuer à tout le monde.
- **Ce code**, bien sûr, n'est pas exécutable lui-même, car l'ordinateur ne comprend que le binaire.
- ce schéma pour comprendre comment cela fonctionne :



Créer un programme

- Le code en langage intermédiaire (CIL) correspond au programme qu'on a distribuer. Sous Windows, il prend l'apparence d'un .exe comme les programmes habituels, mais il ne contient en revanche pas de binaire.
- Lorsqu'on exécute le programme CIL, celui-ci est lu par un autre programme (une machine à analyser les programmes, appelée CLR (JVM)) qui le compile cette fois en vrai programme binaire. Cette fois, le programme peut s'exécuter
- **Rq:**
  - Cela offre beaucoup de souplesse au développeur. Le code en langage intermédiaire (CIL) peut être distribué à tout le monde.
  - Il suffit d'avoir installé la machine CLR sur son ordinateur, qui peut alors lire les programmes en C# et les compiler à la volée en binaire.
- **Avantage :** le programme est toujours adapté à l'ordinateur sur lequel il tourne.

Framework .NET

- On a commencé par parler du C# qui était une brique du framework .NET. Il est temps d'en savoir un peu plus sur le framework.
- Un framework est un ensemble de bibliothèques, de modules et d'outils préconçus qui fournissent une structure et des fonctionnalités communes pour simplifier le processus de développement d'applications.
- Le framework .NET est un framework créé par Microsoft en 2002, en même temps que le C#, qui est principalement dédié à la réalisation d’applications fonctionnant dans des environnements Microsoft. Nous pourrons par exemple réaliser des programmes qui fonctionnent sous Windows, ou bien des sites web ou encore des applications qui fonctionnent sur téléphone mobile, etc.

Résumé

- **Le C#** est un langage de programmation permettant d'utiliser le framework .NET. C'est le langage phare de Microsoft.
- **Le framework .NET** est une énorme boîte à fonctionnalités permettant la création d'applications.
- **Le C#** permet de développer des applications de toutes sortes, exécutables par le CLR qui traduit le MSIL en binaire.

Environnement de développement

Visual Studio :

- L'IDE (Environnement de Développement Intégré) le plus utilisé pour le C#.
- Fonctionnalités : complétion de code, débogage, gestion de projets, tests intégrés.
- Disponible en version Community (gratuite), Professional et Enterprise.

Visual Studio Code :

- Un éditeur de code léger et extensible.
- Nécessite l'installation de l'extension C# pour prendre en charge le développement en C#.
- Idéal pour des projets plus petits ou pour ceux qui préfèrent un outil plus simple.

Les bases de la programmation

- Variables et types de données
- Opérateurs
- Structures de contrôle

les Types de Données

En C#, les types de données définissent le type de valeurs que peut stocker une variable. Les types de données fondamentaux en C# comprennent les types numériques, les types booléens et les types de texte. Comprendre ces types est essentiel pour manipuler les données efficacement dans vos programmes.

- Types Numériques
- Types Booléens
- Type de Texte

Types Numériques

Les types numériques représentent les nombres entiers et les nombres à virgule flottante

a. Type Int

Le type int est utilisé pour stocker des nombres entiers comme des âges, des nombres de comptes, etc. Il peut être positif ou négatif, mais il ne peut pas stocker de nombres avec une partie décimale.

Exemple :

```
int age = 30;
```

b. Type Float

Le type float est utilisé pour stocker des nombres à virgule flottante sur 32 bits.

Exemple :

```
float poids = 65.5;
```

c. Type Double

Le type double est utilisé pour stocker des nombres à virgule flottante sur 64 bits avec une précision accrue par rapport au type float.

Exemple :

```
double pi = 3.14159;
```

d. Type Long

Le type long est utilisé pour stocker de grands nombres entiers signés sur 64 bits.

Exemple :

```
long populationMondiale = 7800000000;
```

.

Types Booléens

Le type bool est utilisé pour représenter les valeurs de vérité, comme vrai (true) ou faux (false). Par exemple, vous pourriez utiliser bool pour vérifier si une condition est vraie ou fausse dans votre programme.

Exemple :

```
bool estVrai = true;
```

Type de Texte

Le type string est utilisé pour stocker du texte. Vous pouvez l'utiliser pour stocker des mots, des phrases ou même des symboles. Par exemple, vous pourriez utiliser string pour stocker le nom d'une personne.

Exemple :

```
string message = "Bonjour, monde !";
```

Variables et types de données

- La déclaration des variables et des constantes est une étape fondamentale dans la programmation en C#. Elle permet de réserver de l'espace en mémoire pour stocker des données et d'attribuer des noms significatifs à ces emplacements.
- Les variables peuvent être modifiées pendant l'exécution du programme, tandis que les constantes ont des valeurs immuables qui ne peuvent pas être changées après leur initialisation

Déclaration des Variables

En C#, les variables doivent être déclarées avant d'être utilisées. La syntaxe générale pour déclarer une variable est la suivante :

type nomVariable;

- **type**: C'est le type de données de la variable (int, float, double, string, etc.).
- **nomVariable**: C'est le nom que vous donnez à la variable pour l'identifier.

Déclaration des Variables

Type de données	Description	Exemple
int	Entier	int age = 25;
string	Chaîne de caractères	string name = "Alice";
bool	Booléen (vrai/faux)	bool isActive = true;
double	Nombre à virgule flottante (précision double)	double price = 19.99;
char	Caractère unique	char grade = 'A';

Déclaration et initialisation

```
int age; // Déclaration

âge = 25; // Initialisation

string name = "Alice"; // Déclaration et initialisation en une ligne
```

Déclaration des Constantes

Les constantes sont des valeurs immuables qui ne peuvent pas être modifiées après leur initialisation. En C#, les constantes sont déclarées à l'aide du mot-clé const. Voici la syntaxe :

const type nomConstante = valeur;

- **type:** Le type de données de la constante.
- **nomConstante:** Le nom de la constante.
- **valeur:** La valeur de la constante, qui doit être assignée lors de la déclaration.

Exemple de déclaration de constante :

const double Pi = 3.14;

const int MoisDansUneAnnee = 12;

Opérateurs

Arithmétiques :

Opérateur	Description	Exemple	Résultat
`+`	Addition	`5 + 3`	`8`
`-`	Soustraction	`5 - 3`	`2`
`*`	Multiplication	`5 * 3`	`15`
`/`	Division	`15 / 3`	`5`
`%`	Modulo (reste)	`5 % 3`	`2`
`**`	Exponentiation	`2 ** 3`	`8`

Opérateurs

Comparaison :

Opérateur	Description	Exemple	Résultat
`==`	Égal à (valeur)	`5 == 5`	`true`
`===`	Égal à (valeur et type)	`5 === "5"`	`false`
`!=`	Différent de (valeur)	`5 != 3`	`true`
`!==`	Différent de (valeur/type)	`5 !== "5"`	`true`
`>`	Supérieur à	`5 > 3`	`true`
`<`	Inférieur à	`5 < 3`	`false`
`>=`	Supérieur ou égal à	`5 >= 5`	`true`
`<=`	Inférieur ou égal à	`5 <= 3`	`false`

Opérateurs

Affectation:

Opérateur	Exemple	Équivalent à	Résultat
`=`	`$a = 5`	—	Assigne 5 à `$a`
`+=`	`$a += 5`	`$a = $a + 5`	Ajoute 5 à `$a`
`-=`	`$a -= 5`	`$a = $a - 5`	Soustrait 5 de `$a`
`*=`	`$a *= 5`	`$a = $a * 5`	Multiplie `$a` par 5
`/=`	`$a /= 5`	`$a = $a / 5`	Divise `$a` par 5
`%=`	`$a %= 5`	`$a = $a % 5`	Modulo de `$a` par 5

Opérateurs

Logiques :

Opérateur	Description	Exemple	Résultat
`&&`	ET logique	`(true && false)`	`false`
`	`		`	OU logique
`!`	NON logique (négation)	`!true`	`false`
`and`	ET logique (faible priorité)	`(true and false)`	`false`
`or`	OU logique (faible priorité)	`(true or false)`	`true`

Présentation générale de la structure d'un programme C#

Méthode Principale :

Chaque programme C# doit contenir une méthode Main, qui est le point d'entrée du programme. C'est là que l'exécution du programme commence. Voici comment cela ressemble :

```
class Program
{
    static void Main(string[] args)
    {
        // Code à exécuter au démarrage du programme
       // Appel méthode , créer des objets ….
    }
}
```

Présentation générale de la structure d'un programme C#

- **static void Main(string[] args)** est la signature de la méthode principale d'une application C#. Cette méthode est statique, ne retourne pas de valeur, s'appelle Main, et peut accepter des arguments en ligne de commande sous forme de tableau de chaînes de caractères.

- La directive **using System**; en C# est une instruction utilisée pour importer l'espace de noms System. L'espace de noms System est l'un des espaces de noms les plus fondamentaux dans le framework .NET et contient de nombreuses classes et fonctionnalités de base nécessaires pour développer des applications C#.

Exercice 1 : Programme de bienvenue

Objectif :

Créer un programme qui demande à l'utilisateur son nom et affiche un message de bienvenue.

Explication :

- Utilisez Console.ReadLine() pour lire l'entrée de l'utilisateur.
- Concaténer la chaîne de caractères pour afficher le message.

Exercice 2

Objectif :

Calcul le le reste de la division entre deux nombres entiers

Explication :

- Utilisez Console.ReadLine() pour lire l'entrée de l'utilisateur.
- choisir deux nombres(a/b) entier et et afficher le reste de la division
  - b!=0

Structures de contrôle

les conditions permettent de contrôler le flux d'exécution d'un programme en fonction de la véracité d'une expression. Elles sont essentielles pour prendre des décisions dans votre code. Voici une petite introduction aux principales structures conditionnelles en C#

La condition if

La condition if-else

La condition else-if

L'opérateur ternaire

La condition switch

Conditions imbriquées

Opérateurs logiques

Structures de contrôle

La condition if La structure if permet d'exécuter un bloc de code **seulement si une condition est vraie**

Syntaxe

```
if (condition)
{
    // Bloc de code à exécuter si la condition est vraie
}
```

Exemple

```
int age = 18;

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

Structures de contrôle

La condition if-else La structure if-else permet d'exécuter un bloc de code si la condition est vraie, et un autre bloc si elle est fausse

Exemple:

```
int age = 16;

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

Structures de contrôle

La condition else-if : La structure else-if permet de tester plusieurs conditions successivement

Exemple:

```
int temperature = 25;
if
(température >= 30)

{
Console.WriteLine("Il fait très chaud ! ☀️");
}
 else if
 (température >= 20)
{
 Console.WriteLine("Le temps est agréable. 😊");
}
 else
 {
 Console.WriteLine("Il fait froid ! ❄️");
}
```

Structures de contrôle

L'opérateur ternaire :L'opérateur ternaire est une manière concise d'écrire une condition if-else sur une seule ligne.

Syntaxe

```
variable = (condition) ? valeurSiVrai : valeurSiFaux;
```

Exemple:

```
int age = 20;
string statut = (age >= 18) ? "Majeur" : "Mineur";
Console.WriteLine(statut); // Affiche "Majeur"
```

Structures de contrôle

Conditions imbriquée :Vous pouvez imbriquer des conditions pour créer des logiques plus complexes..

```
int age = 20;
bool aUnPermis = true;

if (age >= 18)
{
    if (aUnPermis)
    {
        Console.WriteLine("Vous pouvez conduire.");
    }
    else
    {
        Console.WriteLine("Vous devez obtenir un permis.");
    }
}
else
{
    Console.WriteLine("Vous êtes trop jeune pour conduire.");
}
```

Structures de contrôle

Opérateurs logiques: Les opérateurs logiques permettent de combiner plusieurs conditions

Exemple :

```
int age = 25;
bool aUnPermis = true;

if (age >= 18 && aUnPermis)
{
    Console.WriteLine("Vous pouvez conduire.");
}
else
{
    Console.WriteLine("Vous ne pouvez pas conduire.");
}
```

Introduction aux boucles

Les boucles sont des structures de contrôle qui répètent un bloc de code plusieurs fois jusqu'à ce qu'une condition spécifique soit remplie. En C#, il existe principalement trois types de boucles :

- **Boucle for**
- **Boucle while**
- **Boucle do-while**
- **Boucle foreach**

Les Boucles

Boucle for : Répète un bloc de code un nombre spécifié de fois.

```
for (initialisation; condition; itération)
{
    // Bloc de code à répéter
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////
for (int i = 0; i < 5; i++)
{
    Console.WriteLine(i);
}
```

Les Boucles

while : Répète un bloc de code tant qu'une condition est vraie.

```
while (condition)
{
    // Bloc de code à répéter
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int i = 0;
while (i < 5)
{
    Console.WriteLine(i);
    i++;
}
```

Les Boucles

La boucle do-while : C'est une variante de la boucle while, mais elle garantit que le bloc de code est exécuté au moins une fois, même si la condition est fausse dès le début.

```
do
{
    // Bloc de code à répéter
}
while (condition);
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int i = 0;
do
{
    Console.WriteLine(i);
    i++;
}
while (i < 5)
```

Les Boucles

Boucle foreach en C# : La boucle foreach est utilisée pour itérer à travers les éléments d'une collection sans avoir besoin d'indiquer explicitement l'index ou la position de chaque élément. Elle est souvent utilisée lorsque vous devez parcourir tous les éléments d'une collection sans vous soucier de la façon dont elle est implémentée en interne.

```
foreach (type élément in collection)
{
    // Bloc de code à exécuter pour chaque élément
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
string[] noms = { "Alice", "Bob", "Charlie", "David" };

foreach (string nom in noms)
{
    Console.WriteLine(nom);
}
```

Définition d'une méthode

Une méthode est un bloc de code qui effectue une tâche spécifique. Elle peut être appelée à plusieurs endroits dans un programme, ce qui permet de réutiliser du code.

Une méthode peut :

- Prendre des paramètres (données d'entrée).
- Retourner une valeur (résultat).

Définition d'une méthode

En C#, la définition d'une méthode suit la syntaxe suivante :

```
<modificateur> <type de retour> <nom de la méthode>(<paramètres>)
{
   // Corps de la méthode
}
```

<modificateur> : Facultatif, spécifie la portée et le comportement de la méthode (public, private, static, etc.).
<type de retour> : Spécifie le type de données que la méthode renvoie (void si elle ne renvoie rien).
<nom de la méthode> : Le nom de la méthode, utilisé pour l'appeler depuis d'autres parties du programme.

<paramètres> : Facultatif, spécifie les valeurs que la méthode prend en entrée.

Modificateur

- **Private** :signifie que la méthode est accessible uniquement à l'intérieur de la classe où elle est définie.

- **Public** :signifie que la méthode est accessible de n'importe où dans le programme, y compris à l'extérieur de la classe où elle est définie.

- **Static** :signifie que la méthode appartient à la classe elle-même plutôt qu'à une instance spécifique de la classe.

Méthode

❖ Exemple d'une méthode simple qui prend deux nombres en entrée, les additionne et renvoie le résultat :

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

- **`public`** est un modificateur d'accès qui rend la méthode accessible de l'extérieur de la classe.
- **`int`** est le type de retour, indiquant que la méthode renvoie un entier.
- **`Additionner`** est le nom de la méthode**.**
- **`(int a, int b)`** sont les paramètres d'entrée de la méthode.
- Le corps de la méthode effectue l'addition des deux nombres et renvoie la somme.

return ;est utilisé dans les méthodes pour spécifier la valeur que la méthode doit renvoyer à l'endroit où elle a été appelée

Appel d'une Méthode

- Une fois qu'une méthode est définie, vous pouvez l'appeler depuis d'autres parties de votre programme pour exécuter le code qu'elle contient. Pour appeler une méthode, utilisez simplement son nom suivi de parenthèses contenant les arguments nécessaires.
- Reprenons l'exemple précédent et appelons la méthode Additionner :

```
int resultat = Additionner(5, 3);
Console.WriteLine(resultat); // Affiche : 8
```

Dans cet exemple, **Additionner**(5, 3) appelle la méthode Additionner avec les arguments 5 et 3, et le résultat de l'addition est stocké dans la variable resultat. La méthode renvoie ensuite 8, qui est affiché à l'écran.

Exercice : Condition

Vous devez créer un programme qui :

- Demande à l'utilisateur de saisir son âge.
- Affiche la catégorie d'âge en utilisant différentes structures conditionnelles.

Catégories d'âge :

- Enfant : 0 à 12 ans
- Adolescent : 13 à 17 ans
- Jeune adulte : 18 à 25 ans
- Adulte : 26 à 64 ans
- Sénior : 65 ans et plus

Exercice : Boucle

Calcul de la somme des nombres de 1 à 10

Objectif :

- Écrire un programme qui calcule la somme des nombres de 1 à 10 en utilisant une boucle

Explication :

- Initialiser une variable sum à 0.
- Utilisez une boucle pour parcourir les nombres de 1 à 10.
- Ajoutez chaque nombre à sum.

Utilisation de méthodes

Écrivez une méthode CalculerMoyenne qui prend trois nombres en paramètres et retourne leur moyenne.

- Appelez cette méthode depuis Main et affichez le résultat.