



# ECT 330 : DÉVELOPPEMENT AVANCÉ D'APPLICATIONS INTERNET

---

AYENA Adébayer

# QUI SUIS - JE ?

AYENA Adébayer (adebayer.ayena@ipnetinstitute.com / bayorayena@gmail.com / 00228 92 07 79 45)

- ✓ Première année de Licence à la Faculté des Sciences (FDS) de l'Université de Lomé, parcours Mathématiques (2013 – 2014) ;
- ✓ Licence Professionnelle au Centre Informatique et de Calcul (CIC) de l'Université de Lomé, Option Génie Logiciel (2014 - 2017) ;

# QUI SUIS - JE ?

- ✓ Oracle Certified Associate, Java SE 8 Programmer (Juin 2019) ;
- ✓ Oracle Certified Professional, Java SE 8 Programmer (Septembre 2019) ;
- ✓ Oracle Database SQL Certified Associate (Avril 2021).



## **i. OBJECTIFS**

## i. OBJECTIFS

- ❑ Il s'agit d'un cours de programmation axé sur les technologies Internet avancées, telles que la conception d'applications trois tiers, les transactions, la création de composants et les Services Web (Web Services).

## **ii. CONCEPTS TECHNOLOGIQUES COUVERTS PAR LE COURS**

## ii. CONCEPTS TECHNOLOGIQUES COUVERTS PAR LE COURS

- ❑ Transactions LINQ Introduction à Entity Framework (DB First) ;
- ❑ Gestion d'état et mise en cache ;
- ❑ Pages maîtres, contrôles utilisateur et liaison de données ;
- ❑ Technologies de Web Services (REST, SOAP, J2EE, etc.) ;
- ❑ Composants ASP.NET et « Séparation des problèmes NuGet ;
- ❑ Authentification / autorisation ASP.NET.

### **iii. PLAN DU COURS**



## iii. PLAN DU COURS

- ❑ **Partie 1** : Introduction à .NET
- ❑ **Partie 2** : Développement d'applications ASP.NET
- ❑ **Partie 3** : Création de meilleurs formulaires Web
- ❑ **Partie 4** : Utilisation des données
- ❑ **Partie 5** : Sécurité du site Web
- ❑ **Partie 6** : ASP.NET avancé



## **iv. INTRODUCTION**

## iv. INTRODUCTION

- ❑ ASP.NET est la plate-forme de Microsoft pour le développement d'applications Web.
- ❑ En utilisant ASP.NET, vous pouvez créer des boutiques e-commerce, sites de portails basés sur les données et à peu près tout ce que vous pouvez trouver sur Internet.

## iv. INTRODUCTION

- ❑ Pour maîtriser ASP.NET, vous devez apprendre à utiliser un outil de conception avancé (Visual Studio), une boîte à outils d'objets (le Framework .NET) et un langage de programmation orienté objet (tel que C#).
- ❑ Commencer ASP.NET 4.5 en C# suppose que vous souhaitiez maîtriser ASP.NET, en commençant par les bases.

## iv. INTRODUCTION

- ❑ A partir de ce cours, vous développerez vos connaissances jusqu'à ce que vous compreniez les concepts, les techniques et les bonnes pratiques d'écriture applications Web sophistiquées.
- ❑ À la fin, vous trouverez que ASP.NET vous permet de relever des défis qui sont tout simplement hors de portée sur de nombreuses autres plates-formes.



# **PARTIE 1. INTRODUCTION A .NET**

# PARTIE 1. INTRODUCTION A .NET

□ Chap. 1 : Présentation de .NET

□ Chap. 2 : Le langage C#

□ Chap. 3 : Types, objets et espaces de noms (Namespaces)

# CHAP 1. PRÉSENTATION DE .NET



# CHAP 1. PRÉSENTATION DE .NET

- ❑ Le Web existe maintenant depuis environ deux décennies.
- ❑ Pendant ce temps, l'apparence et le fonctionnement des sites Web ont changé dramatiquement. La façon dont les gens créent des sites Web a également évolué.
- ❑ Aujourd'hui, les pages Web peuvent être écrites à la main (peut-être à l'aide d'un outil de conception tel qu'Adobe Dreamweaver), ou ils peuvent être programmés à l'aide de l'un des nombreuses puissantes plates-formes.

# CHAP 1. PRÉSENTATION DE .NET

- ❑ ASP.NET est la boîte à outils de programmation Web de Microsoft.
- ❑ Il fait partie de .NET, un cluster de technologies qui sont conçus pour aider les développeurs à créer une variété d'applications.
- ❑ Les développeurs peuvent utiliser le Framework .NET pour créer des applications Windows, des services qui s'exécutent silencieusement en arrière-plan et même des outils en ligne de commande.

# CHAP 1. PRÉSENTATION DE .NET

- ❑ Les développeurs écrivent le code dans l'un des nombreux langages .NET de base, tels que C#, qui est le langage que nous utiliserons dans ce cours.
- ❑ Dans ce chapitre, nous examinerons les technologies qui sous-tendent .NET.
- ❑ Tout d'abord, nous allons jeter un coup d'œil à l'historique du développement Web et découvrir pourquoi le Framework .NET a été créé.

# CHAP 1. PRÉSENTATION DE .NET

□ Ensuite, nous allons voir un aperçu de haut niveau des parties de .NET et voir comment ASP.NET 4.5 s'intègre dans l'image.

## 1.1. EVOLUTION DU DÉVELOPPEMENT WEB

## 1.1. EVOLUTION DU DÉVELOPPEMENT WEB

- ❑ Internet a vu le jour à la fin des années 1960 comme une expérience.
- ❑ Son objectif était de créer un réseau d'informations vraiment résilient, qui pourrait supporter la perte de plusieurs ordinateurs sans empêcher les autres de communiquer.
- ❑ Poussé par des scénarios de catastrophe potentiels (comme une attaque nucléaire), le département américain de la Défense fournit le financement initial.

## 1.1. EVOLUTION DU DÉVELOPPEMENT WEB

- ❑ Le premier Internet était principalement limité aux établissements d'enseignement et aux entrepreneurs de la défense.
- ❑ Il a prospéré en tant qu'outil de collaboration universitaire, permettant aux chercheurs du monde entier de partager des informations.
- ❑ Au début des années 1990, des modems ont été créés qui pourraient fonctionner sur les lignes téléphoniques existantes, et Internet a commencé à s'ouvrir à utilisateurs commerciaux.

## 1.1. EVOLUTION DU DÉVELOPPEMENT WEB

□ En 1993, le premier navigateur HTML a été créé et la révolution Internet a commencé.





## 1.1.1.1. HTML DE BASE

## 1.1.1.1. HTML DE BASE

- ❑ Il serait difficile de décrire les premiers sites Web comme des applications Web.
- ❑ Au lieu de cela, la première génération de sites Web ressemblait davantage à des brochures, constituées principalement de pages HTML fixes qui devaient être mises à jour manuellement.

## 1.1.1.1. HTML DE BASE

- ❑ Une page HTML de base ressemble un peu à un document de traitement de texte: elle contient un contenu formaté qui peut s'afficher sur votre ordinateur, mais il ne fait rien.
- ❑ L'exemple suivant montre le HTML à sa plus simple forme, avec un document qui contient un titre et une seule ligne de texte :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample Web Page</title>
  </head>
  <body>
    <h1>Sample Web Page Heading</h1>
    <p>This is a sample web page.</p>
  </body>
</html>
```

## 1.1.1.1. HTML DE BASE

- ❑ Tout document HTML respectable devrait commencer par un doctype, un code spécial qui indique quelle saveur de HTML suit.
- ❑ Aujourd'hui, le meilleur choix est le doctype polyvalent suivant, qui a été introduit avec HTML5 mais fonctionne même avec les navigateurs les plus anciens :

```
<!DOCTYPE html>
```

## 1.1.1.1. HTML DE BASE

- ❑ Le reste du document HTML contient le contenu réel.
- ❑ Un document HTML a deux types de contenu: le texte et les éléments (ou balises) qui indiquent au navigateur comment le formater.
- ❑ Les balises sont facilement reconnaissables, car ils sont entourés par des chevrons (<>).

## 1.1.1.1. HTML DE BASE

- ❑ HTML définit des balises pour différents niveaux de titres, paragraphes, hyperliens, mise en forme en italique et en gras, lignes horizontales, etc.
- ❑ Par exemple, `<h1> Du texte </h1>` utilise l'élément `<h1>`.
- ❑ Cet élément indique au navigateur d'afficher du texte dans le style Titre 1, qui utilise une grande police en gras.

## 1.1.1.1. HTML DE BASE

- ❑ De même, `<p>` Ceci est un exemple de page Web. `</p>` crée un paragraphe avec une ligne de texte.
- ❑ L'élément `<head>` regroupe les informations d'en-tête et inclut l'élément `<title>` avec le texte qui apparaît dans la fenêtre du navigateur, tandis que l'élément `<body>` regroupe le contenu réel du document affiché dans la fenêtre du navigateur.



## 1.1.1.1. HTML DE BASE

- ❑ La figure 1-1 montre cette page HTML simple dans un navigateur.
- ❑ Pour le moment, il ne s'agit que d'un fichier fixe (nommé SampleWebPage.html) qui contient juste du contenu HTML.
- ❑ Il n'a aucune interactivité, ne nécessite pas de serveur Web, et ne peut certainement pas être considérée comme une application Web.



**Figure 1-1.** Ordinary HTML

## 1.1.2. FORMULAIRES HTML

## 1.1.2. FORMULAIRES HTML

- ❑ HTML 2.0 a introduit la première graine de programmation Web avec une technologie appelée formulaires HTML.
- ❑ Les formulaires HTML développent le HTML afin qu'il inclut non seulement des balises de mise en forme, mais également des balises pour les widgets graphiques ou les contrôles.
- ❑ Ces contrôles incluent des ingrédients courants tels que des listes déroulantes, des zones de texte et des boutons.

## 1.1.2. FORMULAIRES HTML

□ Voici un exemple de page Web créé avec des contrôles de formulaire HTML :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample Web Page</title>
  </head>
  <body>
    <form>
      <input type="checkbox" />
      This is choice #1<br />
      <input type="checkbox" />
      This is choice #2<br /><br />
      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

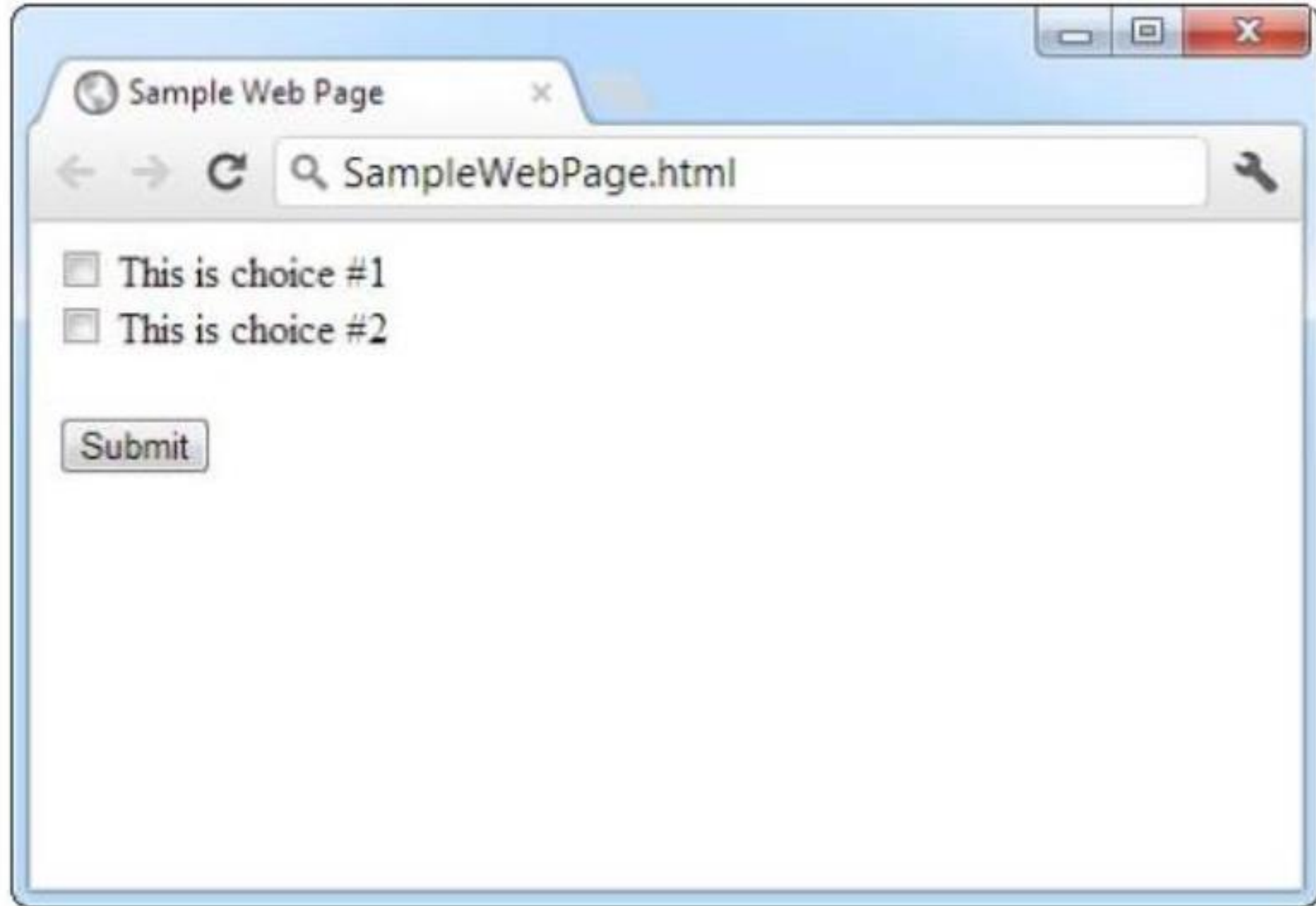
## 1.1.2. FORMULAIRES HTML

- ❑ Dans un formulaire HTML, tous les contrôles sont placés entre les balises `<form>` et `</form>`.
- ❑ L'exemple précédent comprend deux cases à cocher (représentées par l'élément `<input type = "checkbox" />`) et un bouton (représenté par l'élément `<input type = "submit" />`).
- ❑ L'élément `<br />` ajoute un saut de ligne entre les lignes.

## 1.1.2. FORMULAIRES HTML

□ Dans un navigateur, cette page ressemble à la figure 1-2.





**Figure 1-2.** *An HTML form*

## 1.1.2. FORMULAIRES HTML

- ❑ Les formulaires HTML permettent aux développeurs Web de concevoir des pages d'entrée standard.
- ❑ Lorsque l'utilisateur clique sur le bouton « Submit » sur la page illustrée à la Figure 1-2, toutes les données des contrôles d'entrée (dans ce cas, les deux cases à cocher) sont corrigées ensemble dans une longue chaîne de texte et envoyé au serveur Web.

## 1.1.2. FORMULAIRES HTML

- ❑ Côté serveur, une application personnalisée reçoit et traite les données.
- ❑ En d'autres termes, si l'utilisateur coche une case à cocher ou entre du texte, l'application trouve à ce sujet une fois le formulaire soumis.

## 1.1.2. FORMULAIRES HTML

- ❑ Étonnamment, les contrôles qui ont été créés pour les formulaires HTML il y a plus de dix ans sont toujours les fondations que vous utiliserez pour créer des pages ASP.NET dynamiques !
- ❑ La différence est le type d'application qui s'exécute côté serveur.

## 1.1.2. FORMULAIRES HTML

- ❑ Dans le passé, lorsque l'utilisateur cliquait sur un bouton sur une page de formulaire, les informations pouvaient être envoyées par e-mail à un compte défini ou envoyées à une application sur le serveur qui a utilisé la passerelle commune difficile Norme d'interface (CGI).
- ❑ Aujourd'hui, vous travaillerez avec la plate-forme ASP.NET beaucoup plus performante et élégante.



### 1.1.3. ASP.NET

### 1.1.3. ASP.NET

- ❑ ASP.NET est avant tout conçu comme une plate-forme de programmation côté serveur.
- ❑ Cela signifie que tous les codes ASP.NET s'exécutent sur le serveur Web.
- ❑ Lorsque le code ASP.NET finit de s'exécuter, le serveur Web envoie à l'utilisateur le dernier résultat: une page HTML ordinaire qui peut être affichée dans n'importe quel navigateur.

### 1.1.3. ASP.NET

□ ASP.NET utilise la programmation côté serveur pour éviter plusieurs problèmes :

- **Isolation:** le code côté client ne peut pas accéder aux ressources côté serveur. Par exemple, une application côté client n'a pas de moyen simple de lire un fichier ou d'interagir avec une base de données sur le serveur (du moins pas sans rencontrer des problèmes de sécurité et de compatibilité du navigateur).



### 1.1.3. ASP.NET

- **Sécurité:** les utilisateurs finaux peuvent afficher le code côté client. Et une fois que les utilisateurs malveillants ont compris comment une application fonctionne, ils peuvent souvent la falsifier.
- **Clients légers:** dans le monde d'aujourd'hui, les appareils Web tels que les tablettes et les smartphones sont partout. Ces appareils ont généralement une sorte de navigation Web intégrée, mais ils peuvent ne pas prendre en charge les plates-formes de programmation côté client telles que Flash ou Silverlight.

### 1.1.3. ASP.NET

- ❑ Cependant, il est important de comprendre un fait fondamental.
- ❑ Quelles que soient les capacités du navigateur, le code C# que vous écrivez est toujours exécuté sur le serveur.



## 1.2. LE FRAMEWORK .NET

## 1.2. LE FRAMEWORK .NET

□ Comme vous l'avez déjà appris, le Framework .NET est en réalité un cluster de plusieurs technologies:

- **Les langages .NET:** ils incluent Visual Basic, C#, F# et C++, bien que tiers les développeurs en ont créé des centaines d'autres.
- **Le Common Language Runtime (CLR):** il s'agit du moteur qui exécute tous les programmes .NET et fournit des services automatiques pour ces applications, comme la vérification de la sécurité, gestion de la mémoire et optimisation.

## 1.2. LE FRAMEWORK .NET

- ***La bibliothèque de classes du Framework .NET:*** la bibliothèque de classes rassemble des milliers de fonctionnalités prédéfinies que vous pouvez « insérer » dans vos applications. Ces fonctionnalités sont parfois organisées en ensembles de technologies, comme ADO.NET (la technologie pour la création d'applications de base de données) et Windows Presentation Foundation (WPF, la technologie pour créer des interfaces utilisateur de bureau).

## 1.2. LE FRAMEWORK .NET

- **ASP.NET:** il s'agit du moteur qui héberge les applications Web que vous créez avec .NET, et prend en charge presque toutes les fonctionnalités de la bibliothèque de classes du Framework .NET. ASP.NET également comprend un ensemble de services spécifiques au Web, tels que l'authentification sécurisée et le stockage de données.

## 1.2. LE FRAMEWORK .NET

- ***Visual Studio***: cet outil de développement facultatif contient un riche ensemble de productivités et fonctionnalités de débogage. Visual Studio inclut le Framework .NET complet, vous n'avez pas besoin de le télécharger séparément.

## 1.2. LE FRAMEWORK .NET

□ Dans le reste de ce chapitre, vous examinerez rapidement les ingrédients qui composent le Framework .NET



## **CHAP 2. LE LANGAGE C#**

## CHAP 2. LE LANGAGE C#

- ❑ Avant de pouvoir créer une application ASP.NET, vous devez choisir un langage .NET dans lequel la programmer.
- ❑ VB et C# sont des langages modernes et puissants, et vous ne vous tromperez pas en utilisant l'un ou l'autre pour coder vos pages Web.
- ❑ Souvent, le choix est simplement une question de préférence personnelle ou de votre environnement de travail.

## CHAP 2. LE LANGAGE C#

- ❑ Par exemple, si vous avez déjà programmé dans un langage utilisant une syntaxe de type C (par exemple, Java), vous serez probablement le plus à l'aise avec C#.
- ❑ Ou si vous avez passé quelques heures à écrire des macros Microsoft Excel dans VBA, vous préférerez peut-être le style naturel de Visual Basic.
- ❑ De nombreux développeurs maîtrisent les deux.

## CHAP 2. LE LANGAGE C#

- ❑ Ce chapitre présente un aperçu du langage C#.
- ❑ Vous découvrirez les types de données que vous pouvez utiliser, les opérations que vous pouvez effectuer et le code dont vous aurez besoin pour définir les fonctions, les boucles et la logique conditionnelle.
- ❑ Ce part du principe que vous avez déjà programmé et que vous connaissez déjà la plupart de ces concepts – vous il suffit de voir comment ils sont mis en œuvre en C#.

## 2.1. LES LANGAGES DE .NET

## 2.1. LES LANGAGES DE .NET

- ❑ Le Framework .NET est fourni avec deux langages de base couramment utilisés pour créer des applications ASP.NET: C# et VB.
- ❑ Ces langages sont, dans une large mesure, fonctionnellement équivalents.
- ❑ Microsoft a travaillé dur pour éliminer les conflits de langages dans le Framework .NET Framework.

## 2.1. LES LANGAGES DE .NET

- ❑ Ces batailles ralentissent l'adoption, détournent l'attention du noyau de fonctionnalités du Framework, et il est difficile pour la communauté des développeurs de résoudre les problèmes ensemble et de partager des solutions.
- ❑ Selon Microsoft, choisir de programmer en C# au lieu de VB est juste un choix de style de vie et n'affectent pas les performances, l'interopérabilité, l'ensemble des fonctionnalités ou le temps de développement de vos applications.

## 2.1. LES LANGAGES DE .NET

❑ Étonnamment, cette affirmation ambitieuse est essentiellement vraie.



## **2.2. QUELQUES TYPES D'APPLICATIONS ON PEUT CRÉER AVEC C#**

## 2.2. QUELQUES TYPES D'APPLICATIONS ON PEUT CRÉER AVEC C#

- ❑ Les applications informatiques peuvent être de beaucoup de sortes, par exemple une application Windows, comme un logiciel de traitement de texte ou une calculatrice ou encore un jeu de cartes.
- ❑ On les appelle également des clients lourds.
- ❑ Il est également possible de développer des applications web, comme un site d'e-commerce, un intranet, ...

## 2.2. QUELQUES TYPES D'APPLICATIONS ON PEUT CRÉER AVEC C#

- ❑ Nous pourrions accéder à ces applications grâce à un navigateur internet que l'on appelle un client léger.
- ❑ Toujours grâce à un navigateur internet, nous pourrions développer des clients riches.
- ❑ Ce sont des applications qui se rapprochent d'une application Windows mais qui fonctionnent dans un navigateur.

## 2.2. QUELQUES TYPES D'APPLICATIONS ON PEUT CRÉER AVEC C#

- ❑ Bien d'autres types d'applications peuvent être écrites avec le C#, citons encore le développement d'applications mobiles sous Windows phone, d'applications pour tablettes, de jeux ou encore le développement de web services ...
- ❑ Nous verrons un peu plus en détail dans ce cours comment réaliser de telles applications.

## 2.2. QUELQUES TYPES D'APPLICATIONS ON PEUT CRÉER AVEC C#

□ Chacun de ces domaines nécessite un tutoriel entier pour être complètement traité, aussi nous nous initierons à ces domaines sans aller trop loin non plus.

## 2.3. COMPILATION DU CODE C#

## 2.3. COMPILATION DU CODE C#

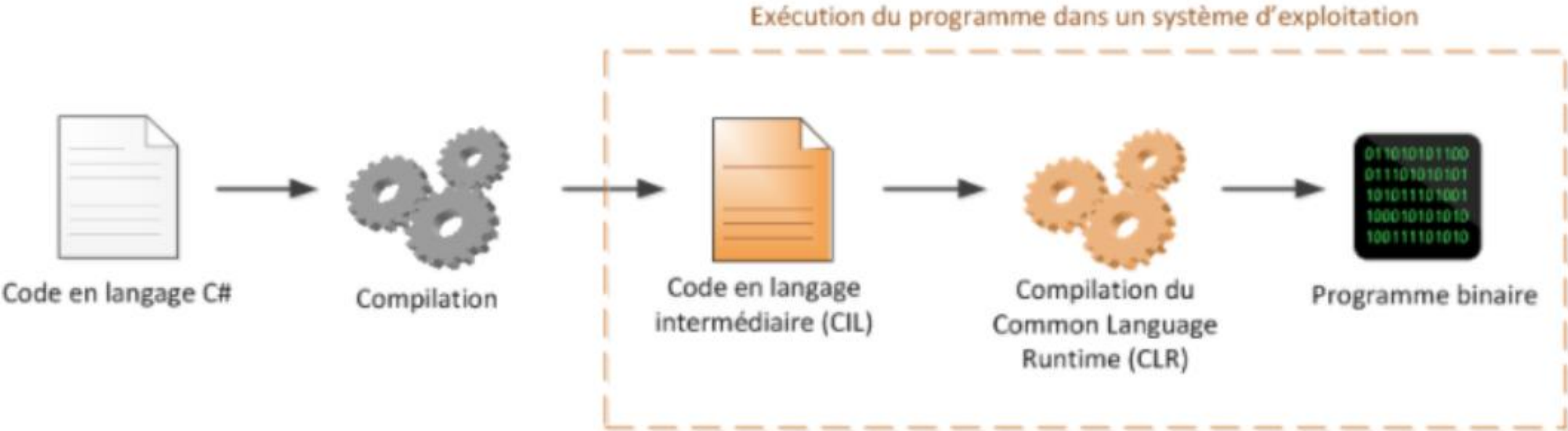
- ❑ Les langages récents, comme le C# et le Java, résolvent le problème de compatibilité à l'OS tout en ajoutant de nombreuses fonctionnalités appréciables au langage, ce qui permet de réaliser des programmes beaucoup plus efficacement.
- ❑ La compilation en C# ne donne pas un programme binaire, contrairement au C et au C++.

## 2.3. COMPILATION DU CODE 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.
- ❑ Regardez bien ce schéma pour comprendre comment cela fonctionne :



## 2.3. COMPILATION DU CODE C#



## 2.3. COMPILATION DU CODE C#

- ❑ Le code en langage intermédiaire (CIL) correspond au programme que vous allez 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) qui le compile cette fois en vrai programme binaire. Cette fois, le programme peut s'exécuter.

## 2.3. COMPILATION DU CODE C#

- ❑ 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.

## **2.4. LE FRAMEWORK .NET ET LE LANGAGE C#**

## 2.4. LE FRAMEWORK .NET ET LE LANGAGE C#

- ❑ Pour simplifier, on peut dire qu'un Framework est une espèce de grosse boîte à fonctionnalités qui va nous permettre de réaliser des applications informatiques.
- ❑ 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.

## 2.4. LE FRAMEWORK .NET ET LE LANGAGE C#

- ❑ Nous pourrions par exemple réaliser des programmes qui fonctionnent sous Windows, ou bien des sites web ou encore des applications qui fonctionnent sur téléphone mobile, etc.
- ❑ Disons que la réalisation d'une application informatique, c'est un peu comme un chantier.

## 2.4. LE FRAMEWORK .NET ET LE LANGAGE C#

- ❑ Il est possible de construire différentes choses, comme une maison, une piscine, une terrasse, etc.
- ❑ Pour réaliser ces constructions, nous allons avoir besoin de matériaux, comme des briques, de la ferraille, etc.
- ❑ Certains matériaux sont communs à toutes les constructions (fer, vis, ...) et d'autres sont spécifiques à certains domaines (pour construire une piscine, je vais avoir besoin d'un liner par exemple).

## 2.4. LE FRAMEWORK .NET ET LE LANGAGE C#

- ❑ On peut voir le Framework .NET comme ces matériaux, c'est un ensemble de composants que l'on devra assembler pour réaliser notre application.
- ❑ Certains sont spécifiques pour la réalisation d'applications web, d'autres pour la réalisation d'applications Windows, etc.
- ❑ Pour réaliser un chantier, nous allons avoir besoin d'outils pour manipuler les matériaux.



## 2.4. LE FRAMEWORK .NET ET LE LANGAGE C#

- ❑ Qui envisagerait de visser une vis avec les doigts ou de poser des parpaings sans les coller avec du mortier ?
- ❑ C'est la même chose pour une application informatique, pour assembler notre application, nous allons utiliser un langage de programmation : le C#.
- ❑ C'est tout ce qu'il y a à savoir pour l'instant, nous reviendrons un peu plus en détail sur le Framework .NET dans les chapitres suivants.

## 2.4. LE FRAMEWORK .NET ET LE LANGAGE C#

□ Pour l'heure, il est important de retenir que c'est grâce au langage de programmation C# et grâce aux composants du Framework .NET que nous allons pouvoir développer des applications informatiques.

## **2.5. CRÉATION D'UN PROJET AVEC VISUAL STUDIO POUR WINDOWS DESKTOP**

## 2.5. CRÉATION D'UN PROJET AVEC VISUAL STUDIO POUR WINDOWS DESKTOP

- ❑ Dans ce chapitre nous allons faire nos premiers pas avec le C#.
- ❑ Nous allons dans un premier temps installer et découvrir les outils qui nous seront nécessaires pour réaliser des applications informatiques avec le C#.
- ❑ Nous verrons comment démarrer avec ces outils et à la fin de ce chapitre, nous serons capables de créer un petit programme qui affiche du texte simple et nous aurons commencé à nous familiariser avec l'environnement de développement.

## 2.5. CRÉATION D'UN PROJET AVEC VISUAL STUDIO POUR WINDOWS DESKTOP

□ Il faut bien commencer par les bases, mais vous verrez comme cela peut être gratifiant d'arriver enfin à faire un petit quelque chose.

## **2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT**

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

- ❑ Pour reprendre la métaphore du chantier, on peut dire qu'il va également nous manquer un chef de chantier.
- ❑ Il n'est pas forcément nécessaire en théorie, mais dans la pratique il se révèle indispensable pour mener à bien son chantier.
- ❑ Ce chef de chantier c'est en fait l'outil de développement.
- ❑ Il va nous fournir les outils pour orchestrer nos développements.

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

□ C'est entre autres :

- Un puissant éditeur ;
- Un compilateur ;
- Un environnement d'exécution.



## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

- ❑ **L'éditeur de texte** va nous servir à créer des fichiers contenant des instructions en langage C#.
- ❑ **Le compilateur** va servir à transformer ces fichiers en une suite d'instructions compréhensibles par l'ordinateur, comme nous l'avons déjà vu.
- ❑ **Le moteur d'exécution** va permettre de faire les actions informatiques correspondantes (afficher une phrase, réagir au clic de la souris, etc.), c'est le CLR dont on a déjà parlé.

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

- ❑ Enfin, il peut être utile d'avoir une base de données.
- ❑ Nous y reviendrons plus en détail dans un autre cours, mais la base de données est un endroit où seront stockées les données de notre application.
- ❑ C'est un élément indispensable à mesure que l'application grandit.
- ❑ Mais pour l'instant, nous n'en avons pas besoin.

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

- ❑ Nous avons donc besoin de notre chef de chantier, l'outil de développement.
- ❑ C'est un logiciel qui va nous permettre de créer des applications et qui va nous fournir les outils pour orchestrer nos développements.
- ❑ La gamme de Microsoft est riche en outils professionnels de qualité pour le développement, notamment grâce à Visual Studio.

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

- ❑ Notez que cet outil de développement se nomme également un IDE pour « Integrated Development Environment » ce qui signifie « Environnement de développement intégré ».
- ❑ Pour apprendre et commencer à découvrir l'environnement de développement, Microsoft propose gratuitement Visual Studio dans sa version Express, aujourd'hui dans sa version Community.

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

- ❑ C'est une version allégée de l'environnement de développement qui permet de faire plein de choses, mais avec des outils en moins par rapport aux versions payantes.
- ❑ Rassurez-vous, ces versions gratuites sont très fournies et permettent de faire tout ce dont on a besoin pour apprendre le C# et suivre ce cours.

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

❑ Pour réaliser des applications d'envergure, il pourra cependant être judicieux d'investir dans l'outil complet et ainsi bénéficier de fonctionnalités complémentaires qui permettent d'améliorer, de faciliter et d'industrialiser les développements.

## 2.5.1. PRÉPARATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT

- ❑ Téléchargement + Installation de Visual Studio.
- ❑ Après l'installation et un peu de configuration automatique, le logiciel s'ouvre sur la page de démarrage de Visual Studio :

# Visual Studio 2022

## Ouvrir les éléments récents

Quand vous utilisez Visual Studio, tous les projets, dossiers ou fichiers que vous ouvrez s'affichent ici pour vous permettre d'y accéder rapidement.

Vous pouvez épingler tout ce que vous ouvrez fréquemment pour l'afficher toujours en tête de liste.

## Démarrer



### Cloner un dépôt

Obtenir du code à partir d'un dépôt en ligne, par exemple GitHub ou Azure DevOps



### Ouvrir un projet ou une solution

Ouvrir un projet ou un fichier .sln Visual Studio local



### Ouvrir un dossier local

Naviguer parmi du code et le modifier dans n'importe quel dossier



### Créer un projet

Choisir un modèle de projet avec génération de modèles automatique de code pour bien démarrer

Continuer sans code →



## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

- ❑ Commençons par créer un nouveau projet en cliquant sur le lien « Créer un projet... » sur la page d'accueil de Visual Studio.
- ❑ Cette commande est également accessible via le menu Fichier > Nouveau > Projet.
- ❑ Un projet va contenir les éléments de ce que l'on souhaite réaliser. Cela peut être par exemple une application web, une application Windows, etc ...

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

- ❑ Commençons par créer un nouveau projet en cliquant sur le lien « Créer un projet... » sur la page d'accueil de Visual Studio.
- ❑ Cette commande est également accessible via le menu Fichier > Nouveau > Projet.
- ❑ Un projet va contenir les éléments de ce que l'on souhaite réaliser. Cela peut être par exemple une application web, une application Windows, etc ...

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET


- ❑ Le projet est aussi un container de fichiers et notamment dans notre cas de fichiers en langage C# qui vont permettre de construire ce que l'on souhaite réaliser.
- ❑ Le projet est en fait représenté par un fichier dont l'extension est .csproj.
- ❑ Son contenu décrit les paramètres de configuration correspondant à ce que l'on souhaite réaliser et les fichiers qui composent le projet.

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET


- ❑ Créons donc un nouveau projet.
- ❑ La fenêtre de création de nouveau projet s'ouvre et nous avons plusieurs possibilités de choix.
- ❑ Nous allons dans un premier temps aller dans Visual C# - Windows pour choisir de créer une Application console (figure suivante).

# Créer un projet

## Modèles de projet récents

 Application web ASP.NET (.NET Framework)

Visual Basic

 Application web ASP.NET (.NET Framework)

C#

Rechercher des modèles (Alt+S)



Tout effacer

C#

Windows

Console



Application console

Projet de création d'une application en ligne de commande pouvant s'exécuter sur .NET sur Windows, Linux et macOS

C#

Linux

macOS

Windows

Console



Application console (.NET Framework)

Projet de création d'une application en ligne de commande

C#

Windows

Console

Vous n'arrivez pas à trouver ce que vous cherchez ?  
[Installer plus d'outils et de fonctionnalités](#)

Retour

Suivant

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

- ❑ Ce que nous faisons ici, c'est utiliser ce qu'on appelle un « modèle » (plus couramment appelé par son équivalent anglais : « template ») de création de projet.
- ❑ Si vous naviguez à l'intérieur des différents modèles, vous pourrez constater que Visual Studio for Windows Desktop nous propose des modèles de projets plus ou moins complexes.

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

- ❑ Ces modèles sont très utiles pour démarrer un projet car toute la configuration du projet est déjà faite.
- ❑ Le nombre de modèles peut être différent en fonction de votre version de Visual Studio.



## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

- ❑ L'application Console est la forme de projet pouvant produire une application exécutable la plus simple.
- ❑ Elle permet de réaliser un programme qui va s'exécuter dans la console noire qui ressemble à une fenêtre ms-dos.
- ❑ Dans cette console, nous allons pouvoir notamment afficher du texte simple, etc.

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

□ Ce type de projet est parfait pour démarrer l'apprentissage du C# car il n'y a besoin que de savoir comment afficher du texte pour commencer alors que pour réaliser une application graphique par exemple, il y a beaucoup d'autres choses à savoir.

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

- ❑ En bas de la fenêtre de création de projet, nous avons la possibilité de choisir un nom pour le projet, ici `ConsoleApplication1`.
- ❑ Changeons le nom de notre application, par exemple "`MaPremiereApplication`", dans la zone correspondante que j'ai entouré en rouge.
- ❑ Cliquons sur OK pour valider la création de notre projet.

## 2.5.2. CRÉATION DE NOTRE PREMIER PROJET

□ Visual Studio crée alors pour nous les fichiers composant une application console vide, qui utilise le C# comme langage et que nous avons nommé `MaPremiereApplication`.

### **2.5.3. ANALYSE RAPIDE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT ET DU CODE GÉNÉRÉ**

### 2.5.3. ANALYSE RAPIDE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT ET DU CODE GÉNÉRÉ

- Allons dans l'emplacement où nous avons créé notre projet, renseigné dans la fenêtre de création de l'application console (par exemple `C:\Users\...\Documents\visual studio ...\Projects` ), nous pouvons constater que Visual Studio a créé un répertoire `MaPremiereApplication`, c'est le répertoire pour la solution.
- Dans ce répertoire, nous remarquons notamment un fichier `MaPremiereApplication.sln`.

### 2.5.3. ANALYSE RAPIDE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT ET DU CODE GÉNÉRÉ

- ❑ Si vous ne voyez pas l'extension du fichier, il faut configurer votre explorateur de fichier pour afficher les extensions connues.
- ❑ C'est ce qu'on appelle le fichier de solution ; il s'agit juste d'un container de projets qui va nous permettre de visualiser nos projets dans Visual Studio.

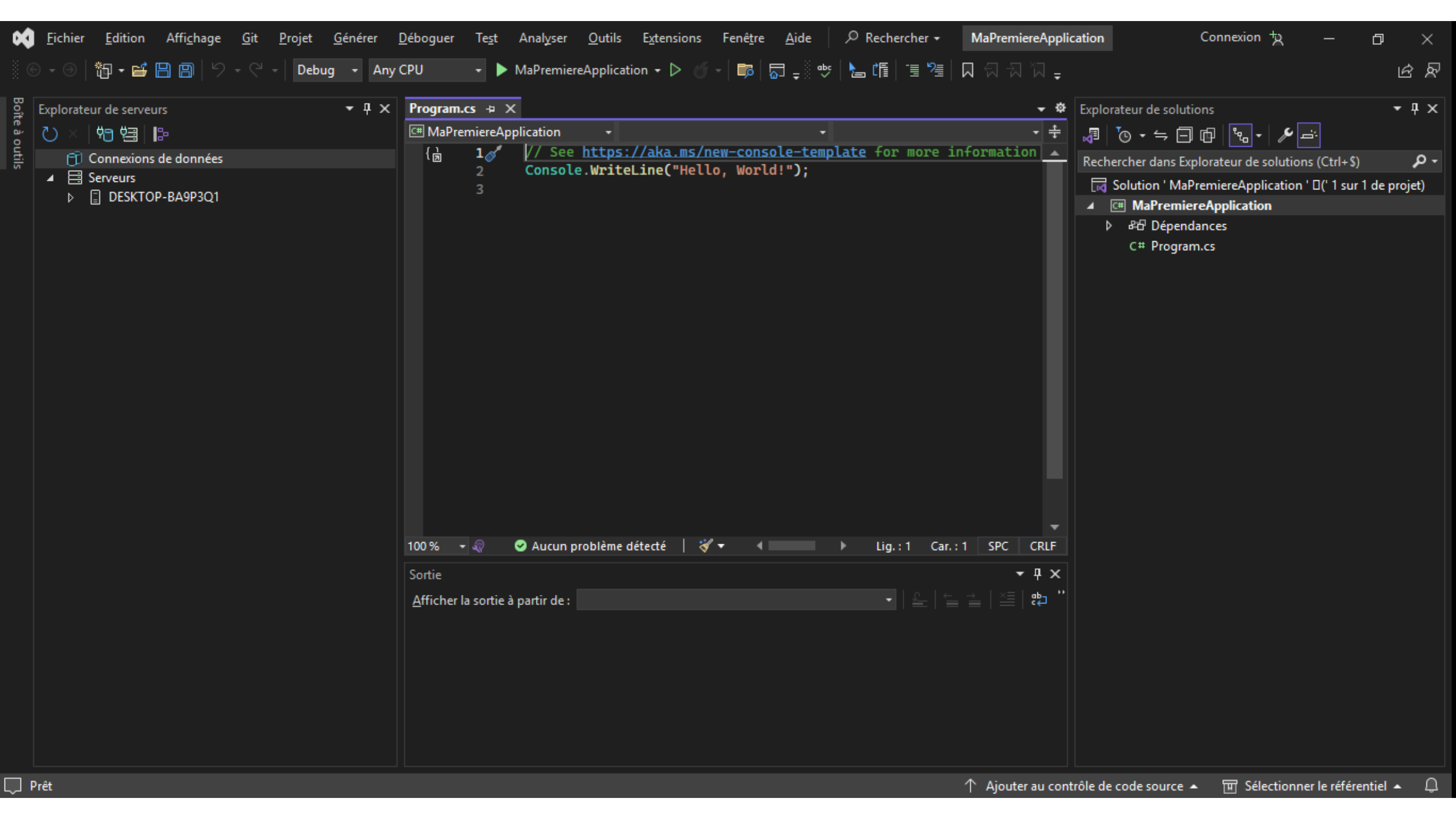
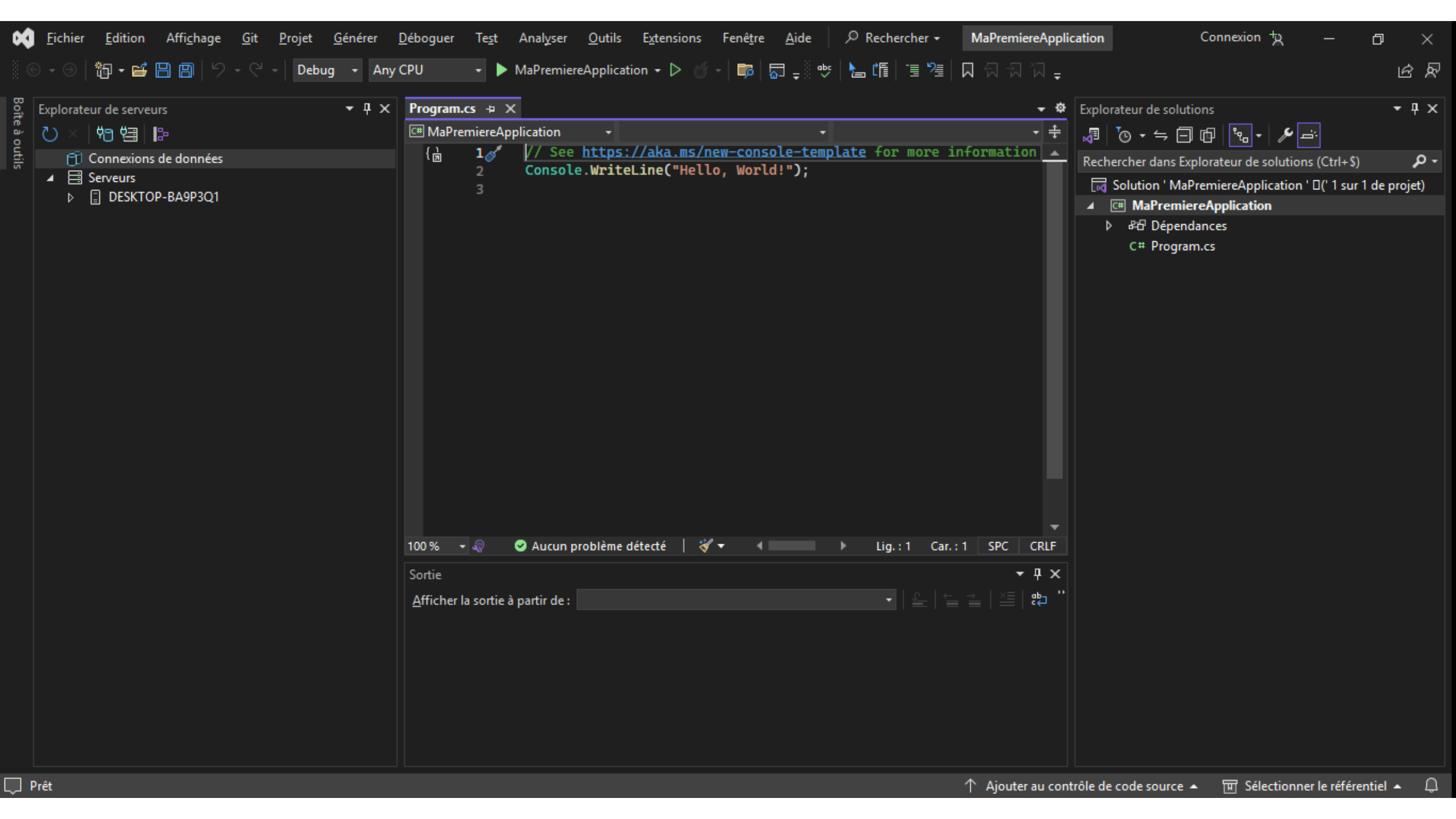
### 2.5.3. ANALYSE RAPIDE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT ET DU CODE GÉNÉRÉ

- En l'occurrence, pour l'instant, nous avons un seul projet dans la solution: l'application `MaPremiereApplication`, que nous retrouvons dans le sous répertoire `MaPremiereApplication` et qui contient notamment le fichier de projet : `MaPremiereApplication.csproj`.
- Le fichier décrivant un projet écrit en C# est préfixé par `cs` (`csproj`).



### 2.5.3. ANALYSE RAPIDE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT ET DU CODE GÉNÉRÉ

- ❑ Il y a encore un fichier digne d'intérêt (pour l'instant) dans ce répertoire, il s'agit du fichier Program.cs.
- ❑ Les fichiers dont l'extension est .cs contiennent du code C#, c'est dans ce fichier que nous allons commencer à taper nos premières lignes de code...
- ❑ Si nous retournons dans l'interface de Visual Studio, nous pouvons retrouver quelque chose comme ça :



#### 2.5.4. ECRITURE DU TEXTE DANS NOTRE APPLICATION

## 2.5.4. ECRITURE DU TEXTE DANS NOTRE APPLICATION

- ❑ Allons donc dans la zone réservée à l'édition de notre fichier Program.cs qui est le fichier contenant le code C# de notre application.
- ❑ Les mots présents dans cette zone sont ce qu'on appelle des instructions de langage.
- ❑ Elles vont nous permettre d'écrire notre programme.

## 2.5.4. ECRITURE DU TEXTE DANS NOTRE APPLICATION

□ Nous reviendrons plus loin sur ce que veulent dire les instructions qui ont été générées par Visual Studio, pour l'instant, rajoutons simplement l'instruction suivante après l'accolade ouvrante :

```
1 Console.WriteLine("Hello World !!");
```

## 2.5.4. ECRITURE DU TEXTE DANS NOTRE APPLICATION

- ❑ Nous venons d'écrire une instruction qui va afficher la phrase "Hello World !!", pour l'instant vous avez juste besoin de savoir ça.
- ❑ Nous étudierons plus en détail ultérieurement à quoi cela correspond exactement.

## **2.5.5. EXECUTION DU PROJET**

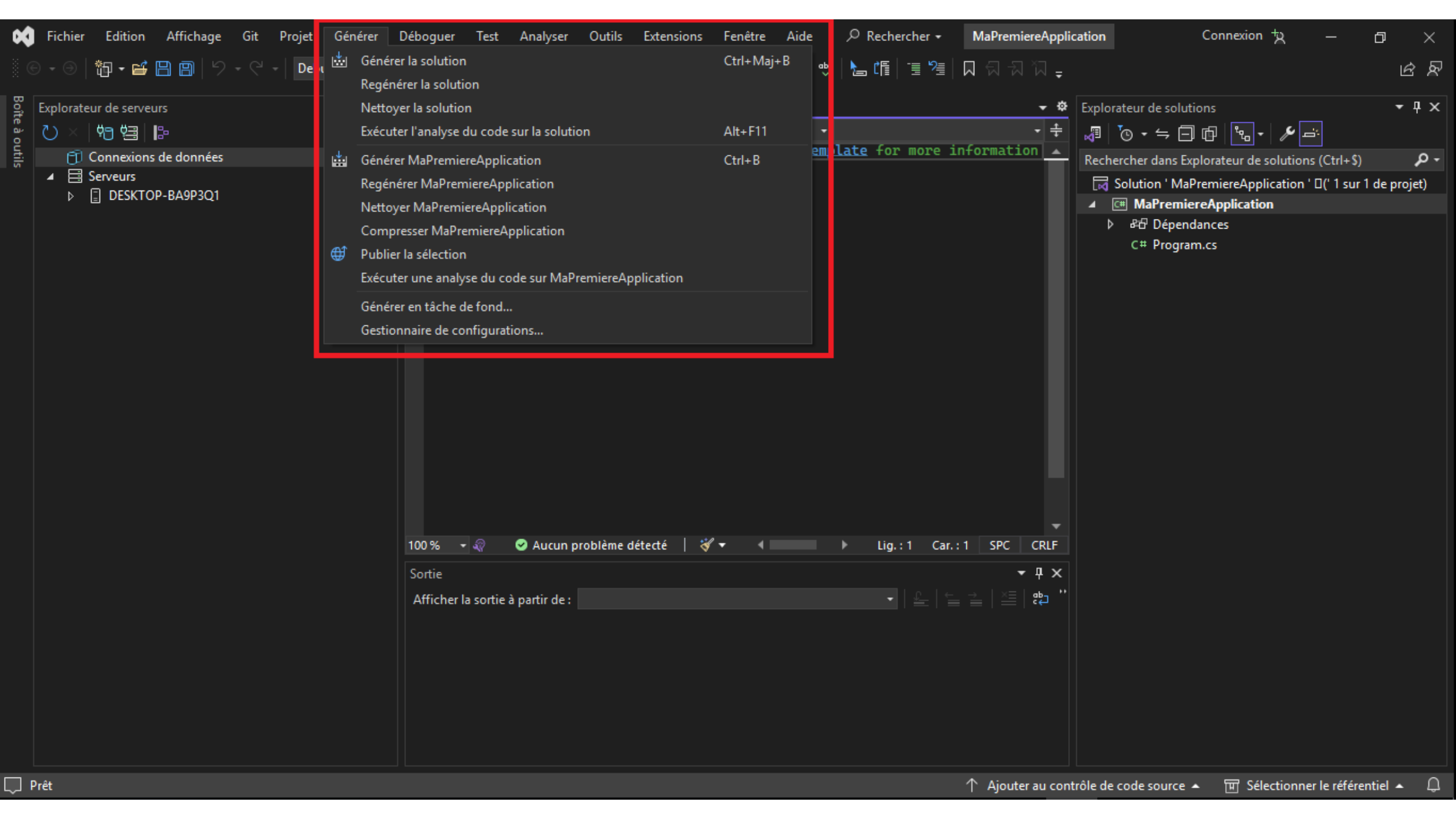
## 2.5.5. EXECUTION DU PROJET

- ❑ Ca y est ! Nous avons écrit notre premier code qui affiche un message très populaire.
- ❑ Mais pour le moment, ça ne fait rien. On veut voir ce que ça donne !
- ❑ La première chose à faire est de transformer le langage C# que nous venons d'écrire en programme exécutable.
- ❑ Cette phase s'appelle la « génération de la solution » sous Visual Studio.



## 2.5.5. EXECUTION DU PROJET

- ❑ On l'appelle souvent la « compilation » ou en anglais le « build ».
- ❑ Pour ce faire, Allez dans le menu « Générer » et cliquez sur « Générer la solution » :



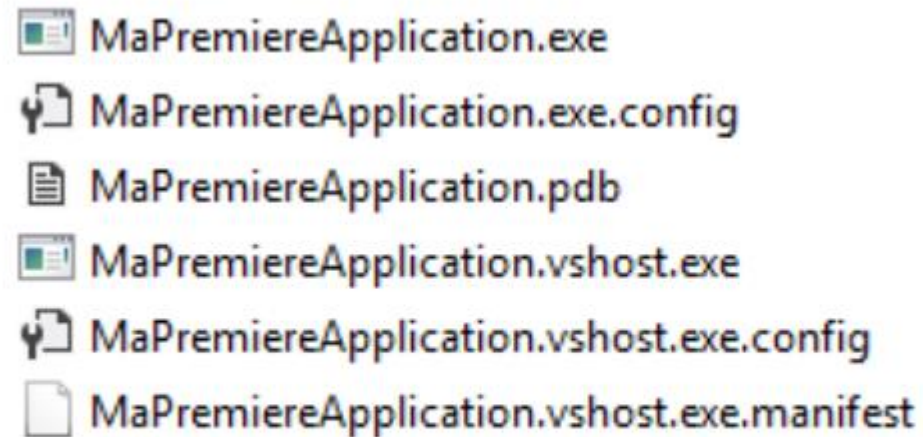
## 2.5.5. EXECUTION DU PROJET

□ Visual Studio lance alors la génération de la solution et on voit dans la barre des tâches en bas à gauche qu'il travaille jusqu'à nous indiquer que la génération a réussi :

## 2.5.5. EXECUTION DU PROJET

- ❑ Vous aurez noté à ce moment-là que la fenêtre de sortie s'affiche pour nous montrer le résultat de la compilation.
- ❑ Si nous allons dans le répertoire contenant la solution, nous pouvons voir dans le répertoire `MaPremiereApplication\MaPremiereApplication\bin\debug` qu'il y a plusieurs fichiers :

## 2.5.5. EXECUTION DU PROJET



---

Fichiers générés pour notre application console

## 2.5.5. EXECUTION DU PROJET

- ❑ Le premier est le fichier exécutable, possédant l'extension .exe, qui est le résultat du processus de génération. Il s'agit bien de notre application.
- ❑ Le second n'est pas utile à connaître pour l'instant, il s'agit de la configuration de l'application.
- ❑ Les fichiers suivants sont des fichiers particuliers qu'il n'est pas utile de connaître pour l'instant, nous allons les ignorer (ils servent pour le débogage de l'application).

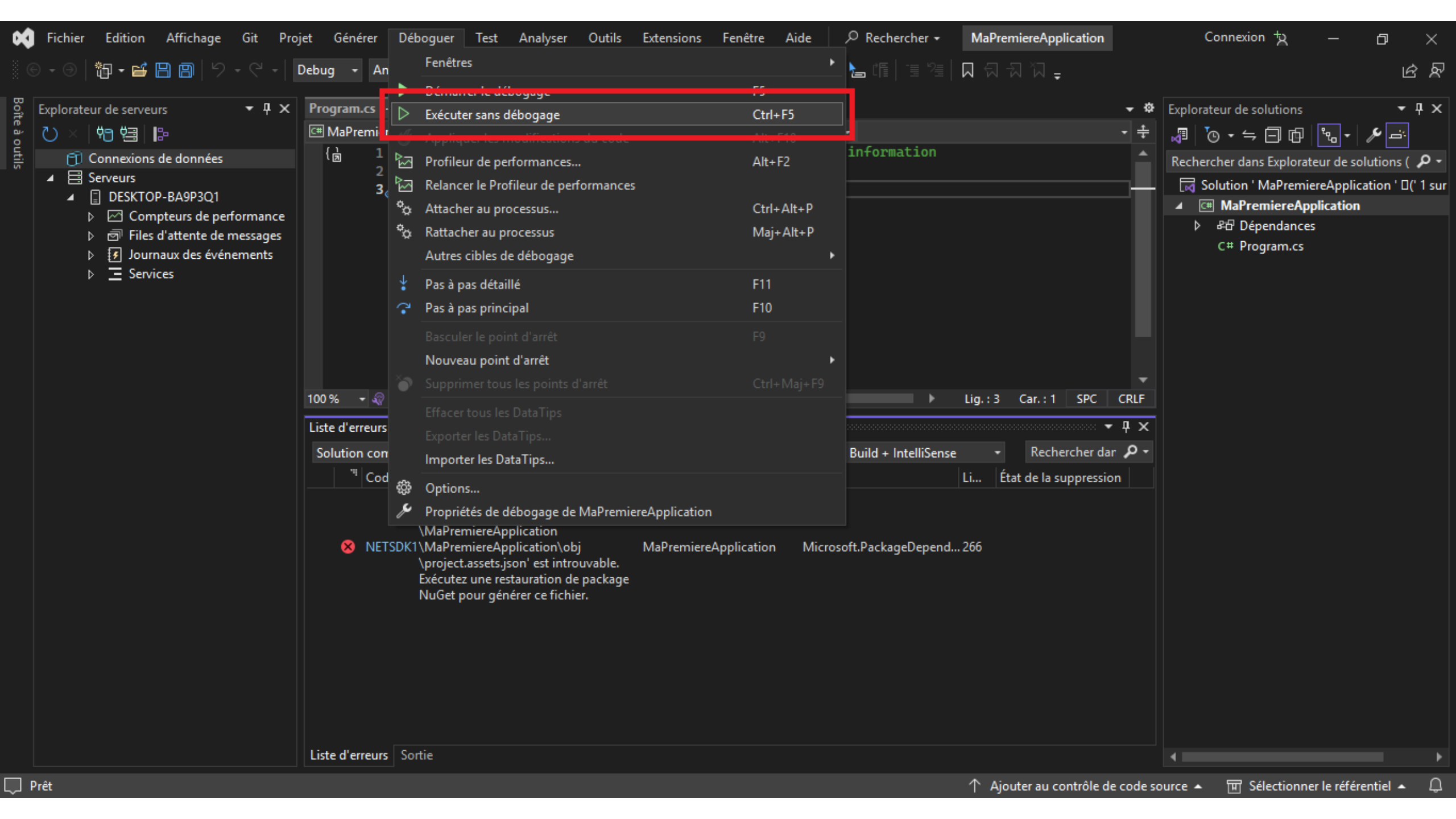
## 2.5.5. EXECUTION DU PROJET

- ❑ Exécutons notre application en lançant le fichier exécutable depuis l'explorateur de fichiers.
- ❑ Déception, nous voyons à peine un truc noirâtre qui s'affiche et qui se referme immédiatement. Que s'est-il passé ?
- ❑ En fait, l'application s'est lancée, a affiché notre message et s'est terminée immédiatement.

## 2.5.5. EXECUTION DU PROJET

- ❑ Et tout ça un brin trop rapidement ... ça ne va pas être pratique tout ça.
- ❑ Heureusement, Visual Studio arrive à la rescousse.
- ❑ Retournons dans notre IDE préféré et allons dans le menu Débogueur pour ensuite choisir l'élément de menu Exécuter sans débogage :





## 2.5.5. EXECUTION DU PROJET

□ La console s'ouvre nous délivrant le message tant attendu :

C:\Windows\system32\cmd.exe

Hello World !!

Appuyez sur une touche pour continuer...

## 2.5.5. EXECUTION DU PROJET

- ❑ Le message est désormais visible car Visual Studio nous demande d'appuyer sur une touche pour que l'application se termine, ce qui nous laisse donc le temps d'apprécier l'exécution de notre superbe programme.
- ❑ Ça y est, notre première application en C#.

## 2.6. LES PRINCIPES DE BASE DU LANGAGE C#

## 2.6. LES PRINCIPES DE BASE DU LANGAGE C#

- ❑ Les nouveaux programmeurs C# sont parfois intimidés par la syntaxe excentrique du langage, qui inclut des caractères tels que des points-virgules (;), des accolades ({}), et des barres obliques inversées (\).
- ❑ Heureusement, une fois que vous devenez des habitués au C#, ces détails se fondront rapidement dans l'arrière-plan.
- ❑ Dans les sections suivantes, vous apprendrez sur quatre principes généraux que vous devez connaître sur C# avant d'apprendre d'autres concepts.

## 2.6.1. SENSIBILITÉ À LA CASSE

## 2.6.1. SENSIBILITÉ À LA CASSE

- ❑ Certaines langues sont sensibles à la casse, tandis que d'autres ne le sont pas.
- ❑ Java, C, C++ et C# sont tous des exemples de langages sensibles à la casse.
- ❑ VB ne l'est pas.



## 2.6.1. SENSIBILITÉ À LA CASSE

- ❑ Cette différence peut frustrer les anciens programmeurs VB qui ne réalisent pas que les mots clés, les variables et les fonctions doivent être saisies avec la casse appropriée.
- ❑ Par exemple, si vous essayez de créer un conditionnel en C# en saisissant `If` au lieu de `if`, votre code ne sera pas reconnu et le compilateur le marquera avec une erreur lorsque vous essayez de compiler votre application.

## 2.6.1. SENSIBILITÉ À LA CASSE

- ❑ C# a également une préférence définie pour les mots minuscules.
- ❑ Mots clés, tels que `if`, `for`, `foreach`, `while`, `typeof`, et ainsi de suite – sont toujours écrits en minuscules.
- ❑ Lorsque vous définissez vos propres variables, il est logique de suivre les conventions utilisées par d'autres programmeurs C# et la bibliothèque de classes du Framework .NET.

## 2.6.2. LES COMMENTAIRES

## 2.6.2. LES COMMENTAIRES

- ❑ Les commentaires sont des lignes de texte descriptif ignorées par le compilateur.
- ❑ C# fournit deux types de commentaires de base.
- ❑ Le premier type est le commentaire sur une seule ligne.

## 2.6.2. LES COMMENTAIRES

□ Dans ce cas, le commentaire commence par deux barres obliques (//) et continue pour toute la ligne courante:

```
// A single-line C# comment.
```

## 2.6.2. LES COMMENTAIRES

□ En option, les programmeurs C# peuvent utiliser les symboles `/*` et `*/` pour indiquer des commentaires sur plusieurs lignes:

```
/* A multiple-line  
   C# comment. */
```

### 2.6.3. FIN DES INSTRUCTIONS

## 2.6.3. FIN DES INSTRUCTIONS

- ❑ C# utilise un point-virgule (;) comme caractère de fin d'instruction.
- ❑ Chaque instruction dans le code C# doit se terminer par un point-virgule, sauf lorsque vous définissez une structure de bloc (des exemples de telles déclarations incluent des méthodes, les instructions conditionnelles et les boucles, qui sont trois types d'ingrédients de code que vous découvrirez plus loin dans ce chapitre).



## 2.6.3. FIN DES INSTRUCTIONS

- ❑ En omettant le point-virgule, vous pouvez facilement diviser une instruction de code sur plusieurs lignes.
- ❑ Vous ne devez juste pas oublier de mettre le point-virgule à la fin de la dernière ligne pour terminer l'instruction.
- ❑ L'extrait d code suivant montre quatre méthodes équivalentes pour effectuer la même opération (en ajoutant trois numéros ensemble):

```
// A code statement on a single line.  
myValue = myValue1 + myValue2 + myValue3;
```

```
// A code statement split over two lines.  
myValue = myValue1 + myValue2 +  
          myValue3;
```

```
// A code statement split over three lines.  
myValue = myValue1 +  
          myValue2 +  
          myValue3;
```

```
// Two code statements in a row.  
myValue = myValue1 + myValue2;  
myValue = myValue + myValue3;
```

## 2.6.3. FIN DES INSTRUCTIONS

- ❑ Comme vous pouvez le voir dans cet exemple, C# vous donne un large éventail de liberté pour diviser votre instruction comme vous le souhaitez.
- ❑ La règle générale est de rendre votre code aussi lisible que possible.
- ❑ Ainsi, si vous avez une longue déclaration, vous pouvez étaler la déclaration sur plusieurs lignes pour qu'elle soit plus facile à lire.

## 2.6.3. FIN DES INSTRUCTIONS

❑ Par contre, si vous avez une complexe instruction de code qui effectue plusieurs opérations à la fois, vous pouvez étaler l'instruction sur plusieurs lignes ou séparez votre logique en plusieurs instructions de code pour la rendre plus claire.

#### 2.6.4. LES BLOCS DE CODES

## 2.6.4. LES BLOCS DE CODES

- ❑ Les langages C#, Java et C reposent tous fortement sur des accolades – des parenthèses avec un peu plus d'attitude: {}.
- ❑ Les accolades regroupent plusieurs instructions de code ensemble.
- ❑ En règle générale, vous regroupez les instructions de code car vous voulez qu'ils soient répétés dans une boucle, exécutés conditionnellement ou regroupés dans une fonction.

## 2.6.4. LES BLOCS DE CODES

- ❑ Ce sont tous des blocs structures, et vous verrez toutes ces techniques dans ce chapitre.
- ❑ Mais dans chaque cas, les accolades jouent le même rôle, ce qui rend C# plus simple et plus concis que les autres langages qui nécessitent une syntaxe différente pour chaque type de la structure du bloc.

```
{  
    // Code statements go here.  
}
```

## 2.7. VARIABLES ET TYPES DE DONNÉES



## 2.7. VARIABLES ET TYPES DE DONNÉES

- ❑ Comme avec tous les langages de programmation, vous gardez une trace des données en C# à l'aide de variables.
- ❑ Les variables peuvent stocker des nombres, texte, dates et heures, et ils peuvent même pointer vers des objets à part entière.
- ❑ Lorsque vous déclarez une variable, vous lui donnez un nom et spécifiez le type de données qu'elle stockera.

## 2.7.1. DÉCLARATION DE VARIABLES

## 2.7.1. DÉCLARATION DE VARIABLES

- ❑ Pour déclarer une variable locale, vous commencez la ligne avec le type de données, suivi du nom que vous souhaitez utiliser.
- ❑ Un dernier point-virgule termine la déclaration.

## 2.7.1. DÉCLARATION DE VARIABLES

```
// Declare an integer variable named errorCode.  
int errorCode;
```

```
// Declare a string variable named myName.  
string myName;
```

## 2.7.2. TYPES DE DONNÉES

## 2.7.2. TYPES DE DONNÉES

- ❑ Chaque langage .NET utilise les mêmes types de données.
- ❑ Différents langages peuvent fournir légèrement de différents noms.
- ❑ Cette conception permet une intégration linguistique.
- ❑ Pour créer ce système de type de données commun, Microsoft a créé un ensemble de types de données de base, qui sont fournis dans la bibliothèque de classes .NET.

## 2.7.2. TYPES DE DONNÉES

□ Le tableau 2-1 répertorie les types de données de base les plus importants.

**Table 2-1.** *Common Data Types*

C# Name	VB Name	.NET Type Name	Contains
byte	Byte	Byte	An integer from 0 to 255.
short	Short	Int16	An integer from -32,768 to 32,767.
int	Integer	Int32	An integer from -2,147,483,648 to 2,147,483,647.
long	Long	Int64	An integer from about -9.2e18 to 9.2e18.
float	Single	Single	A single-precision floating-point number from approximately -3.4e38 to 3.4e38 (for big numbers) or -1.5e-45 to 1.5e-45 (for small fractional numbers).
double	Double	Double	A double-precision floating-point number from approximately -1.8e308 to 1.8e308 (for big numbers) or -5.0e-324 to 5.0e-324 (for small fractional numbers).
decimal	Decimal	Decimal	A 128-bit fixed-point fractional number that supports up to 28 significant digits.



char	Char	Char	A single Unicode character.
string	String	String	A variable-length series of Unicode characters.
bool	Boolean	Boolean	A true or false value.
*	Date	DateTime	Represents any date and time from 12:00:00 AM, January 1 of the year 1 in the Gregorian calendar, to 11:59:59 PM, December 31 of the year 9999. Time values can resolve values to 100 nanosecond increments. Internally, this data type is stored as a 64-bit integer.
*	*	TimeSpan	Represents a period of time, as in ten seconds or three days. The smallest possible interval is 1 <i>tick</i> (100 nanoseconds).
object	Object	Object	The ultimate base class of all .NET types. Can contain any data type or object. (You'll take a much closer look at objects in Chapter 3.)

---

*\* If the language does not provide an alias for a given type, you must use the .NET type name.*

### 2.7.3. ASSIGNATIONS ET INITIALISATIONS

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ Une fois que vous avez déclaré vos variables, vous pouvez leur attribuer librement des valeurs, à condition que ces valeurs aient le bon type de données.
- ❑ Voici le code qui montre ce processus en deux étapes:

### 2.7.3. ASSIGNATIONS ET INITIALISATIONS

```
// Declare variables.  
int errorCode;  
string myName;
```

```
// Assign values.  
errorCode = 10;  
myName = "Matthew";
```

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ Vous pouvez également affecter une valeur à une variable dans la même ligne que vous la déclarez.
- ❑ Cet exemple compresse le précédant quatre lignes de code en deux:

```
int errorCode = 10;  
string myName = "Matthew";
```

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

❑ C# vous protège des erreurs en vous empêchant d'utiliser des variables non initialisées.

❑ Par exemple, le code suivant provoque une erreur lorsque vous essayez de le compiler:

```
int number;           // Number is uninitialized.  
number = number + 1;  // This causes a compile error.
```

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

□ La bonne façon d'écrire ce code est d'initialiser explicitement la variable numérique à une valeur appropriée, telle comme 0, avant de l'utiliser:

```
int number = 0;           // Number now contains 0.  
number = number + 1;      // Number now contains 1.
```

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ C# traite également strictement des types de données.
- ❑ Par exemple, l'instruction de code suivante ne fonctionnera pas telle qu'elle est écrite:

```
decimal myDecimal = 14.5;
```



## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ Le problème est que la valeur littérale 14.5 est automatiquement interprété comme un double et que vous ne pouvez pas convertir un double en décimal sans utiliser la syntaxe de conversion, qui est décrite plus loin dans ce chapitre.
- ❑ Pour contourner ce problème, C# définit quelques caractères spéciaux que vous pouvez ajouter aux valeurs littérales pour indiquer leur type de données afin que une conversion sera nécessaire.

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

□ Ces caractères sont les suivants:

- M (decimal)
- D (double)
- F (float)
- L (long)

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

❑ Par exemple, vous pouvez réécrire l'exemple précédent en utilisant l'indicateur décimal comme suit:

```
decimal myDecimal = 14.5M;
```

❑ Dans cet exemple, une lettre majuscule M est utilisée, mais vous pouvez le remplacer par une lettre minuscule m.

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ Les indicateurs de type de données sont l'un des rares détails qui ne sont pas sensibles à la casse en C#.
- ❑ Fait intéressant, si vous utilisez un code comme celui-ci pour déclarer et initialiser votre variable en une seule étape, et si le compilateur C# peut déterminer le bon type de données en fonction de la valeur que vous utilisez, vous n'avez pas besoin de spécifier les types données.

### 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ Au lieu de cela, vous pouvez utiliser le mot-clé polyvalent « var » à la place du type de données.
- ❑ Cela signifie que la ligne précédente de code équivaut à ceci:

```
var myDecimal = 14.5M;
```

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ Ici, le compilateur se rend compte qu'un type de données « decimal » est le choix le plus approprié pour la variable « myDecimal » et utilise ce type de données automatiquement.
- ❑ Il n'y a pas de différence de performance.
- ❑ La variable « myDecimal » que vous créez à l'aide d'un type de données inféré se comporte exactement de la même manière qu'une variable « myDecimal » créée avec un type de données explicite.

## 2.7.3. ASSIGNATIONS ET INITIALISATIONS

- ❑ En fait, le code de bas niveau généré par le compilateur est identique.
- ❑ La seule différence est que le mot-clé « var » économise un peu de saisie.
- ❑ De nombreux programmeurs C# se sentent mal à l'aise avec le mot clé « var » car il rend le code moins clair.
- ❑ Cependant, le mot-clé « var » est un raccourci plus utile lors de la création d'objets, comme nous le verrons dans le chapitre suivant.

## 2.8. CHAINES ET CARACTÈRES D'ÉCHAPPEMENT



## 2.8. CHAINES ET CARACTÈRES D'ÉCHAPPEMENT

- ❑ C# traite le texte un peu différemment des autres langages tels que VB.
- ❑ Il interprète toute barre oblique inverse (\\) incorporée comme début d'une séquence de caractères spéciaux.
- ❑ Par exemple, \\n signifie ajouter une nouvelle ligne (retour à la ligne).

## 2.8. CHAINES ET CARACTÈRES D'ÉCHAPPEMENT

□ Les plus utiles sont les suivants:

- `\"` (double quote)
- `\n` (new line)
- `\t` (horizontal tab)
- `\\` (backward slash)

## 2.9. OPÉRATIONS SUR LES VARIABLES

## 2.9. OPÉRATIONS SUR LES VARIABLES

- ❑ Vous pouvez utiliser tous les types standard d'opérations sur les variables en C#.
- ❑ Lorsque vous travaillez avec des nombres, vous pouvez utiliser divers symboles mathématiques, comme indiqué dans le tableau 2-2.
- ❑ C# suit l'ordre conventionnel des opérations, effectuant l'exponentiation d'abord, suivis de la multiplication et de la division, puis de l'addition et de la soustraction.

## 2.9. OPÉRATIONS SUR LES VARIABLES

❑ Vous pouvez également contrôler l'ordre par regroupement des sous-expressions entre parenthèses:

```
int number;
```

```
number = 4 + 2 * 3;  
// number will be 10.
```

```
number = (4 + 2) * 3;  
// number will be 18.
```

## 2.9. OPÉRATIONS SUR LES VARIABLES

**Table 2-2.** *Arithmetic Operations*

Operator	Description	Example
+	Addition	$1 + 1 = 2$
-	Subtraction	$5 - 2 = 3$
*	Multiplication	$2 * 5 = 10$
/	Division	$5.0 / 2 = 2.5$
%	Gets the remainder left after integer division	$7 \% 3 = 1$

## 2.9. OPÉRATIONS SUR LES VARIABLES

- ❑ La division peut parfois prêter à confusion en C#.
- ❑ Si vous divisez un entier par un autre entier, C# effectue une division entière.
- ❑ Cela signifie qu'il rejette automatiquement la partie fractionnaire de la réponse et renvoie la partie entière sous forme d'entier.
- ❑ Par exemple, si vous divisez 5 par 2, vous obtiendrez 2 au lieu de 2,5.

## 2.9. OPÉRATIONS SUR LES VARIABLES

- ❑ Par exemple, si vous divisez 5 par 2, vous obtiendrez 2 au lieu de 2,5.
- ❑ La solution consiste à indiquer explicitement que l'un de vos nombres est une valeur fractionnaire.
- ❑ Par exemple, si vous remplacez 5 par 5M, C# traitera le 5 comme une décimale.



## 2.9. OPÉRATIONS SUR LES VARIABLES

- ❑ Si vous remplacez 5 par 5.0, C# le traitera comme un double.
- ❑ D'une manière ou d'une autre, la division renverra la valeur attendue de 2,5.
- ❑ Bien sûr, ce problème ne se produit pas très souvent dans le monde réel, car vous divisez généralement une variable par une autre.
- ❑ Tant que vos variables ne sont pas des entiers, peu importe le nombre qu'ils contiennent.

## 2.9. OPÉRATIONS SUR LES VARIABLES

- ❑ Les opérateurs du tableau 2-2 sont conçus pour manipuler les nombres.
- ❑ Cependant, C# vous permet également d'utiliser l'opérateur d'addition (+) pour joindre deux chaînes:

```
// Join three strings together.  
myName = firstName + " " + lastName;
```

## 2.9. OPÉRATIONS SUR LES VARIABLES

❑ De plus, C# fournit des opérateurs d'affectation de raccourcis spéciaux.

❑ Voici quelques exemples:

```
// Add 10 to myValue. This is the same as myValue = myValue + 10;  
myValue += 10;
```

```
// Multiple myValue by 3. This is the same as myValue = myValue * 3;  
myValue *= 3;
```

```
// Divide myValue by 12. This is the same as myValue = myValue / 12;  
myValue /= 12;
```

## 2.10. OPÉRATIONS MATHÉMATIQUES AVANCÉES SUR LES VARIABLES

## 2.10. OPÉRATIONS MATHÉMATIQUES AVANCÉES SUR LES VARIABLES

- ❑ Dans le passé, chaque langage avait son propre ensemble de mots-clés pour les opérations mathématiques courantes telles que l'arrondi, la racine carrée, etc.
- ❑ Dans les langages .NET, bon nombre de ces mots-clés demeurent.
- ❑ Cependant, vous pouvez également utiliser un outil mathématique centralisé dans une classe qui fait partie du .NET Framework.

## 2.10. OPÉRATIONS MATHÉMATIQUES AVANCÉES SUR LES VARIABLES

- ❑ Cela a pour effet secondaire agréable de garantir que le code que vous utilisez pour effectuer des opérations mathématiques peut facilement être traduit en instructions équivalentes dans n'importe quel langage .NET.
- ❑ Pour utiliser les opérations mathématiques, vous appelez les méthodes de la classe `System.Math`.
- ❑ Ces méthodes sont statiques, ce qui signifie qu'ils sont toujours disponibles et prêts à l'emploi.

## 2.10. OPÉRATIONS MATHÉMATIQUES AVANCÉES SUR LES VARIABLES

- ❑ Le chapitre suivant explore plus en détails la différence entre les membres statiques et les membres d'instance.
- ❑ L'extrait de code suivant montre quelques exemples de calculs que vous pouvez effectuer avec la classe `Math` :

## 2.10. OPÉRATIONS MATHÉMATIQUES AVANCÉES SUR LES VARIABLES

```
double myValue;  
myValue = Math.Sqrt(81);           // myValue = 9.0  
myValue = Math.Round(42.889, 2);   // myValue = 42.89  
myValue = Math.Abs(-10);           // myValue = 10.0  
myValue = Math.Log(24.212);        // myValue = 3.18.. (and so on)  
myValue = Math.PI;                 // myValue = 3.14.. (and so on)
```



## 2.10. OPÉRATIONS MATHÉMATIQUES AVANCÉES SUR LES VARIABLES

- ❑ Les fonctionnalités de la classe `Math` sont trop nombreuses pour être listées ici dans leur intégralité.
- ❑ Les exemples précédents montrent certaines opérations numériques courantes.

## 2.10. OPÉRATIONS MATHÉMATIQUES AVANCÉES SUR LES VARIABLES

□ Pour plus d'informations sur les fonctions trigonométriques et logarithmiques disponibles, reportez-vous aux informations de référence du cours de mathématiques sur le site Web MSDN de Microsoft :

<http://msdn.microsoft.com/library/system.math.aspx>

## 2.11. CONVERSIONS DE TYPE

## 2.11. CONVERSIONS DE TYPE

- ❑ La conversion d'informations d'un type de données à un autre est une tâche de programmation assez courante.
- ❑ Par exemple, vous pouvez récupérer l'entrée de texte d'un utilisateur contenant le nombre que vous souhaitez utiliser pour un calcul.
- ❑ Ou, vous pourriez avoir besoin pour prendre une valeur calculée et la transformer en texte, vous pouvez l'afficher dans une page Web.

## 2.11. CONVERSIONS DE TYPE

- ❑ Les conversions sont de deux types : élargissement et rétrécissement.
- ❑ Les conversions en expansion (élargissement) réussissent toujours.
- ❑ Par exemple, vous pouvez toujours convertir un entier 32 bits en entier 64 bits.
- ❑ Vous n'aurez besoin d'aucun code spécial :

## 2.11. CONVERSIONS DE TYPE

```
int mySmallValue;  
long myLargeValue;  
  
// Get the largest possible value that can be stored as a 32-bit integer.  
// .NET provides a constant named Int32.MaxValue that provides this number.  
mySmallValue = Int32.MaxValue;  
  
// This always succeeds. No matter how large mySmallValue is,  
// it can be contained in myLargeValue.  
myLargeValue = mySmallValue;
```

## 2.11. CONVERSIONS DE TYPE

- ❑ En revanche, les conversions de type « réduction » peut réussir ou échouer, selon les données.
- ❑ Lors de la conversion d'un entier 32 bits en entier 16 bits, vous pourriez rencontrer une erreur si le nombre 32 bits est supérieur à la valeur maximale qui peut être stockée dans le type de données 16 bits.
- ❑ Toutes les conversions restrictives doivent être effectuées explicitement.

## 2.11. CONVERSIONS DE TYPE

- ❑ C# utilise une méthode élégante pour la conversion de type explicite.
- ❑ Pour convertir une variable, il vous suffit de spécifiez le type entre parenthèses avant l'expression que vous convertissez.
- ❑ Le code suivant montre comment modifier un entier 32 bits en entier 16 bits :



## 2.11. CONVERSIONS DE TYPE

```
int count32 = 1000;  
short count16;  
  
// Convert the 32-bit integer to a 16-bit integer.  
// If count32 is too large to fit, .NET will discard some of the  
// information you need, and the resulting number will be incorrect.  
count16 = (short)count32;
```

## 2.11. CONVERSIONS DE TYPE

- ❑ Ce processus s'appelle le casting.
- ❑ Si vous n'utilisez pas de distribution explicite lorsque vous essayez d'effectuer une conversion de « rétrécissement », vous recevrez une erreur lorsque vous tenterez de compiler votre code.
- ❑ Cependant, même si vous effectuez une conversion, vous pourriez toujours vous retrouver avec un problème.

## 2.11. CONVERSIONS DE TYPE

❑ Par exemple, considérez le code montré ici, qui provoque un débordement:

```
int mySmallValue;  
long myLargeValue;  
  
myLargeValue = Int32.MaxValue;  
myLargeValue++;  
  
// This will appear to succeed (there won't be an error at runtime),  
// but your data will be incorrect because mySmallValue cannot  
// hold a value this large.  
mySmallValue = (int)myLargeValue;
```

## 2.11. CONVERSIONS DE TYPE

- ❑ En C#, vous ne pouvez pas utiliser le casting pour convertir des nombres en chaînes, ou vice versa.
- ❑ Dans ce cas, les données ne sont pas simplement déplacées d'une variable à une autre - il doit être traduit dans un format complètement différent.
- ❑ Heureusement, .NET propose un certain nombre de solutions pour effectuer des conversions avancées.

## 2.11. CONVERSIONS DE TYPE

□ Une option consiste à utiliser les méthodes statiques de la classe « Convert », qui prend en charge de nombreux types de données courants tels que les chaînes, les dates et les nombres.

## 2.11. CONVERSIONS DE TYPE

```
string countString = "10";
```

```
// Convert the string "10" to the numeric value 10.  
int count = Convert.ToInt32(countString);
```

```
// Convert the numeric value 10 into the string "10".  
countString = Convert.ToString(count);
```

## 2.11. CONVERSIONS DE TYPE

- ❑ La deuxième étape (transformer un nombre en chaîne) fonctionnera toujours.
- ❑ La première étape (transformer une chaîne en un number) ne fonctionnera pas si la chaîne contient des lettres ou d'autres caractères non numériques, auquel cas une erreur va se produire.
- ❑ Le chapitre 7 décrit comment utiliser la gestion des erreurs pour détecter et neutraliser ce type de problème.

## 2.11. CONVERSIONS DE TYPE

- ❑ La classe « Convert » est une bonne solution polyvalente, mais vous trouverez également d'autres méthodes statiques capables de faire le travail, si vous fouillez dans la bibliothèque de classes .NET.
- ❑ Le code suivant utilise la méthode statique `Int32.Parse()` pour effectuer la même tâche:



## 2.11. CONVERSIONS DE TYPE

```
int count;  
string countString = "10";  
  
// Convert the string "10" to the numeric value 10.  
count = Int32.Parse(countString);
```

## 2.11. CONVERSIONS DE TYPE

- ❑ Vous constaterez également que vous pouvez utiliser des méthodes d'objet pour effectuer certaines conversions de manière un peu plus élégante.
- ❑ La section suivante illustre cette approche avec la méthode `ToString()`.

## 2.12. LES STRUCTURES CONDITIONNELLES

## 2.12. LES STRUCTURES CONDITIONNELLES

- ❑ À bien des égards, la structure conditionnelle : décider de l'action à entreprendre en fonction des entrées de l'utilisateur, des conditions externes ou autres l'information - est au cœur de la programmation.
- ❑ Toute structure conditionnelle commence par une condition: une expression simple qui peut être évaluée à True (vraie) ou False (faux).
- ❑ Votre code peut alors prendre la décision d'exécuter une logique différente en fonction du résultat de la condition.

## 2.12. LES STRUCTURES CONDITIONNELLES

- ❑ Pour construire une condition, vous pouvez utiliser n'importe quelle combinaison de valeurs littérales ou de variables avec des opérateurs logiques.
- ❑ Le Tableau 2-7 liste les opérateurs logiques de base :

**Table 2-7.** *Logical Operators*

Operator	Description
==	Equal to.
!=	Not equal to.
<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.
&&	Logical and (evaluates to true only if both expressions are true). If the first expression is false, the second expression is not evaluated.
	Logical or (evaluates to true if either expression is true). If the first expression is true, the second expression is not evaluated.

## 2.12. LES STRUCTURES CONDITIONNELLES

- ❑ Vous pouvez utiliser tous les opérateurs de comparaison avec tous les types numériques.
- ❑ Avec les types de données chaîne, vous ne pouvez utiliser que les opérateurs d'égalité (== et !=).
- ❑ C# ne prend pas en charge les autres types d'opérateurs de comparaison de chaînes.

## 2.12. LES STRUCTURES CONDITIONNELLES

- ❑ Vous devez utiliser la méthode `String.Compare()`.
- ❑ La méthode `String.Compare()` considère qu'une chaîne est « inférieure à » une autre chaîne si elle apparaîtrait plus tôt dans un tri alphabétique.
- ❑ La valeur de retour de `String.Compare()` vaut 0 si les chaînes correspondent, 1 si la première chaîne fournie est supérieure à la seconde et -1 si la première string est inférieur à la seconde.



## 2.12. LES STRUCTURES CONDITIONNELLES

❑ Voici un exemple :

```
int result;  
result = String.Compare("apple", "attach"); // result = -1  
result = String.Compare("apple", "all");    // result = 1  
result = String.Compare("apple", "apple");  // result = 0  
  
// Another way to perform string comparisons.  
string word = "apple";  
result = word.CompareTo("attach");           // result = -1
```



## **2.12.1. L'INSTRUCTION IF**

## 2.12.1. L'INSTRUCTION IF

- ❑ L'instruction if est le moteur de la structure conditionnelle, capable d'évaluer n'importe quelle combinaison de conditions et de traiter avec des données multiples et différentes.
- ❑ Voici un exemple avec une instruction if qui comporte deux conditions else :

## 2.12. LES STRUCTURES CONDITIONNELLES

```
if (myNumber > 10)
{
    // Do something.
}
else if (myString == "hello")
{
    // Do something.
}
else
{
    // Do something.
}
```

## 2.12.1. L'INSTRUCTION IF

- ❑ Un bloc if peut avoir n'importe quel nombre de conditions.
- ❑ Si vous ne testez qu'une seule condition, vous n'avez pas besoin d'inclure tout autre bloque else.
- ❑ Dans cet exemple, chaque bloc est clairement identifié par les caractères { }.
- ❑ Ceci est une exigence si vous souhaitez écrire plusieurs lignes de code dans un bloc conditionnel.

## 2.12.1. L'INSTRUCTION IF

- ❑ Si votre bloc conditionnel ne nécessite qu'une seule instruction, vous pouvez omettre les accolades.
- ❑ Cependant, ce n'est jamais une mauvaise idée de les conserver, car cela rend votre code clair et sans ambiguïté.

## 2.12.1. L'INSTRUCTION IF

- ❑ Gardez à l'esprit que la structure if correspond au plus à une condition.
- ❑ Par exemple, si myNumber est supérieur à 10, la première condition sera remplie.
- ❑ Cela signifie que le code du premier bloc conditionnel s'exécutera, et les autres conditions ne seront plus évaluées.

## 2.12.1. L'INSTRUCTION IF

- ❑ Le fait que `myString` contienne le texte `bonjour` devient sans importance, car cela la condition ne sera pas évaluée.
- ❑ Si vous souhaitez vérifier les deux conditions, n'utilisez pas de bloc `else`.
- ❑ Vous avez besoin deux `if` blocs indépendants, comme indiqué ici:



## 2.12.1. L'INSTRUCTION IF

```
if (myNumber > 10)
{
    // Do something.
}
if (myString == "hello")
{
    // Do something.
}
```

## **2.1 2.2. L'INSTRUCTION SWITCH**

## 2.12.2. L'INSTRUCTION SWITCH

- ❑ C# fournit également une instruction « switch » que vous pouvez utiliser pour évaluer une seule variable ou expression pour plusieurs valeurs possibles.
- ❑ La seule limitation est que la variable que vous évaluez doit être un type de données basées sur un entier, un booléen, un caractère, une chaîne ou une valeur d'une énumération.
- ❑ Les autres types de données ne sont pas pris en charge.

## 2.1 2.2. L'INSTRUCTION SWITCH

□ Dans le code suivant, chaque cas examine la variable myNumber et teste si elle est égale à un entier:

## 2.12.2. L'INSTRUCTION SWITCH

```
switch (myNumber)
{
    case 1:
        // Do something.
        break;
    case 2:
        // Do something.
        break;
    default:
        // Do something.
        break;
}
```

## 2.12.2. L'INSTRUCTION SWITCH

- ❑ Vous remarquerez que la syntaxe C# hérite de la convention de programmation C/C++, qui exige que chaque branch dans une instruction switch se termine par un mot-clé spécial break.
- ❑ Si vous omettez ce mot-clé, le compilateur vous alertera et refusera de compiler votre application.

## 2.12.2. L'INSTRUCTION SWITCH

- ❑ La seule exception est si vous choisissez d'empiler plusieurs instructions case directement les unes sur les autres sans code intermédiaire.
- ❑ Cela vous permet d'écrire un segment de code qui traite plus d'un cas.
- ❑ Voici un exemple :

## 2.12.2. L'INSTRUCTION SWITCH

```
switch (myNumber)
{
    case 1:
    case 2:
        // This code executes if myNumber is 1 or 2.
        break;
    default:
        // Do something.
        break;
}
```



## 2.12.2. L'INSTRUCTION SWITCH

- ❑ Contrairement à l'instruction if, l'instruction switch se limite à évaluer une seule information à la fois.
- ❑ Cependant, il fournit une syntaxe plus légère et plus claire que l'instruction if lorsque vous devez tester une seule variable.

## **2.13. LES INSTRUCTIONS RÉPÉTITIVES (BOUCLES)**

## 2.13. LES INSTRUCTIONS RÉPÉTITIVES (BOUCLES)

- ❑ Les boucles vous permettent de répéter plusieurs fois un segment de code.
- ❑ C# a trois types de boucles de base.
- ❑ Vous choisissez le type de boucle en fonction du type de tâche que vous devez effectuer.

## 2.13. LES INSTRUCTIONS RÉPÉTITIVES (BOUCLES)

□ Vos choix sont les suivants:

- Vous pouvez effectuer une boucle un certain nombre de fois avec une boucle « for ».
- Vous pouvez parcourir tous les éléments d'une collection de données en utilisant une boucle « foreach ».

## 2.13. LES INSTRUCTIONS RÉPÉTITIVES (BOUCLES)

- Vous pouvez effectuer une boucle tant qu'une certaine condition est vraie avec une boucle « while » ou avec une boucle « do...while ».

## 2.13. LES INSTRUCTIONS RÉPÉTITIVES (BOUCLES)

- ❑ Les boucles for et foreach sont idéales pour parcourir des ensembles de données qui ont des tailles fixes connues.
- ❑ La boucle while est une construction plus flexible qui vous permet de continuer le traitement jusqu'à ce qu'une condition complexe soit remplie.
- ❑ La boucle while est souvent utilisée avec des tâches ou des calculs répétitifs qui n'ont pas un nombre défini d'itérations.

## 2.13.1. LA BOUCLE FOR

## 2.13.1. LA BOUCLE FOR

- ❑ La boucle for est un ingrédient de base dans de nombreux programmes.
- ❑ Il vous permet de répéter un bloc de code un nombre défini de fois, en utilisant un compteur intégré.
- ❑ Pour créer une boucle for, vous devez spécifier une valeur de départ, une valeur de fin et le type d'incrément à chaque itération.
- ❑ Voici un exemple :



### 2.13.1. LA BOUCLE FOR

```
for (int i = 0; i < 10; i++)  
{  
    // This code executes ten times.  
    System.Diagnostics.Debug.Write(i);  
}
```

## 2.13.1. LA BOUCLE FOR

- ❑ Vous remarquerez que la boucle for commence par des parenthèses qui indiquent trois informations importantes.
- ❑ La première partie (`int i = 0`) crée la variable de compteur (`i`) et définit sa valeur initiale (0).
- ❑ La troisième partie (`i++`) incrémente la variable de compteur.
- ❑ Dans cet exemple, le compteur est incrémenté de 1 après chaque itération.

## 2.13.1. LA BOUCLE FOR

- ❑ Cela signifie  $i$  sera égal à 0 pour la première itération, égal à 1 pour la seconde itération, et ainsi de suite.
- ❑ Cependant, vous pouvez ajuster cela afin qu'il décrémente le compteur (ou exécute toute autre opération que vous souhaitez).
- ❑ La partie médiane ( $i < 10$ ) spécifie la condition qui doit être remplie pour que la boucle continue.

## 2.13.1. LA BOUCLE FOR

- ❑ Cette condition est testée au début de chaque passage à travers le bloc.
- ❑ Si `i` est supérieur ou égal à 10, la condition sera évaluée à `False` (faux) et la boucle s'arrêtera.
- ❑ Si vous exécutez le code précédent à l'aide d'un outil tel que Visual Studio, il écrira les nombres suivants :

0 1 2 3 4 5 6 7 8 9

## 2.13.1. LA BOUCLE FOR

- ❑ Il est souvent judicieux de définir la variable de compteur en fonction du nombre d'éléments que vous traitez.
- ❑ Par exemple, vous pouvez utiliser une boucle for pour parcourir les éléments d'un tableau en vérifiant la taille du tableau avant de commencer.
- ❑ Voici le code que vous utiliseriez :

## 2.13.1. LA BOUCLE FOR

```
string[] stringArray = {"one", "two", "three"};

for (int i = 0; i < stringArray.Length; i++)
{
    System.Diagnostics.Debug.Write(stringArray[i] + " ");
}
```

## 2.13.1. LA BOUCLE FOR

❑ Ce code produit la sortie suivante :

one two three

## 2.13.2. LA BOUCLE FOREACH

- ❑ C# fournit également une boucle foreach qui vous permet de parcourir les éléments d'un ensemble de données.
- ❑ Avec une boucle foreach, vous n'avez pas besoin de créer une variable de compteur explicite.
- ❑ Au lieu de cela, vous créez une variable qui représente le type de les données que vous recherchez.



## 2.13.2. LA BOUCLE FOREACH

- ❑ Votre code passera ensuite en boucle jusqu'à ce que vous ayez eu la possibilité de traiter chaque élément de données dans l'ensemble.
- ❑ La boucle foreach est particulièrement utile pour parcourir les données dans les collections et les tableaux.
- ❑ Par exemple, le prochain le segment de code parcourt les éléments d'un tableau en utilisant la boucle foreach.

## 2.13.2. LA BOUCLE FOREACH

□ Ce code a exactement le même effet que l'exemple dans la section précédente, mais c'est un peu plus simple :

## 2.13.2. LA BOUCLE FOREACH

```
string[] stringArray = {"one", "two", "three"};

foreach (string element in stringArray)
{
    // This code loops three times, with the element variable set to
    // "one", then "two", and then "three".
    System.Diagnostics.Debug.Write(element + " ");
}
```

## 2.13.2. LA BOUCLE FOREACH

- ❑ Dans ce cas, la boucle foreach examine chaque élément du tableau et tente de le convertir en chaîne.
- ❑ Donc, la boucle foreach définit une variable de chaîne nommée element.
- ❑ Si vous avez utilisé un autre type de données, vous recevrez une erreur.
- ❑ La boucle foreach a une limitation clé: elle est en lecture seule.

## 2.13.2. LA BOUCLE FOREACH

- ❑ Par exemple, si vous souhaitez parcourir un tableau et modifier les valeurs de ce tableau en même temps, cela ne fonctionnera pas.
- ❑ Voici un exemple de quelques code défectueux :

## 2.13.2. LA BOUCLE FOREACH

```
int[] intArray = {1,2,3};  
foreach (int num in intArray)  
{  
    num += 1;  
}
```

## 2.13.2. LA BOUCLE FOREACH

❑ Dans ce cas, vous devrez vous rabattre sur une boucle for basique avec un compteur.

### **2.13.3. LA BOUCLE WHILE**



## 2.13.3. LA BOUCLE WHILE

- ❑ Enfin, C# prend en charge une boucle while qui teste une condition spécifique avant ou après chaque itération dans la boucle.
- ❑ Quand cette condition prend la valeur False (false), la boucle est arrêtée.
- ❑ Voici un exemple qui effectue dix boucles.
- ❑ Au début de chaque itération, le code évalue si le compteur (i) est inférieur à une limite supérieure (dans ce cas, 10).

### 2.13.3. LA BOUCLE WHILE

□ Si tel est le cas, la boucle effectue une autre itération.

```
int i = 0;
while (i < 10)
{
    i += 1;
    // This code executes ten times.
}
```

## 2.13.3. LA BOUCLE WHILE

- ❑ Vous pouvez également placer la condition à la fin de la boucle en utilisant la syntaxe « do ... while ».
- ❑ Dans ce cas, la condition est testée à la fin de chaque itération dans la boucle :

### 2.13.3. LA BOUCLE WHILE

```
int i = 0;
do
{
    i += 1;
    // This code executes ten times.
}
while (i < 10);
```

## 2.13.3. LA BOUCLE WHILE

❑ Ces deux exemples sont équivalents, sauf si la condition que vous testez est fausse pour commencer :

- la boucle while ignorera complètement le code.
- La boucle do ... while, par contre, exécutera toujours le code à au moins une fois, car il ne teste pas la condition qu'à la fin.

## 2.13.3. LA BOUCLE WHILE

- ❑ Parfois, vous devez quitter une boucle rapidement.
- ❑ En C#, vous pouvez utiliser l'instruction `break` pour quitter tout type de boucle.
- ❑ Vous pouvez également utiliser l'instruction `continue` pour ignorer le reste de la passe en cours, évaluer la condition et (si elle renvoie `True`), commencer l'itération suivante.

## 2.14. LES FONCTIONS

## 2.14. LES FONCTIONS/METHODES

- ❑ Les fonctions sont le bloc de construction le plus basique que vous pouvez utiliser pour organiser votre code.
- ❑ Essentiellement, une fonction est un regroupement d'une ou plusieurs lignes de code.
- ❑ Idéalement, chaque fonction effectuera une tâche logique distincte.



## 2.14. LES FONCTIONS/METHODES

- ❑ En structurant votre code en méthodes, vous simplifiez non seulement votre vie, mais vous facilitez également l'organisation de votre code en classes et vous entrez dans le monde de la programmation orientée objet.
- ❑ La première décision à prendre lors de la déclaration d'une méthode est de savoir si vous souhaitez renvoyer information.

## 2.14. LES FONCTIONS/METHODES

- ❑ Par exemple, une méthode nommée `GetStartTime()` peut renvoyer un objet `DateTime` qui représente l'heure à laquelle une application a été lancée pour la première fois.
- ❑ Une méthode peut renvoyer, au maximum, une donnée.

## 2.14.1. DÉCLARATION DES MÉTHODES

## 2.14.1. DÉCLARATION DES MÉTHODES

- ❑ Lorsque vous déclarez une méthode en C#, la première partie de la déclaration spécifie le type de données de la valeur de retour, et la deuxième partie indique le nom de la méthode.
- ❑ Si votre méthode ne renvoie aucune information, vous devez utiliser le mot-clé `void` au lieu d'un type de données au début de la déclaration.
- ❑ Voici deux exemples: une méthode qui ne renvoie rien et une qui le fait:

```
// This method doesn't return any information.  
void MyMethodNoReturnedData()  
{  
    // Code goes here.  
}
```

```
// This method returns an integer.  
int MyMethodReturnsData()  
{  
    // As an example, return the number 10.  
    return 10;  
}
```

## 2.14.1. DÉCLARATION DES MÉTHODES

- ❑ Notez que le nom de la méthode est toujours suivi de parenthèses.
- ❑ Cela permet au compilateur de reconnaître que c'est une méthode.
- ❑ Dans cet exemple, les méthodes ne spécifient pas leur accessibilité.
- ❑ Il s'agit simplement d'une convention C# courante.
- ❑ Tu es libre d'ajouter un mot-clé d'accessibilité (tel que `public` ou `private`), comme indiqué ici :

## 2.14.1. DÉCLARATION DES MÉTHODES

```
private void MyMethodNoReturnedData()  
{  
    // Code goes here.  
}
```

## 2.14.1. DÉCLARATION DES MÉTHODES

- ❑ L'accessibilité détermine la façon dont les différentes classes de votre code peuvent interagir.
- ❑ Les méthodes privées sont masquées et ne sont disponibles que localement, alors que les méthodes publiques peuvent être appelées par toutes les autres classes de votre application.
- ❑ Pour vraiment comprendre ce que cela signifie, vous devrez lire le chapitre suivant, qui traite l'accessibilité plus en détail.



## 2.14.1. DÉCLARATION DES MÉTHODES

- ❑ Si vous ne spécifiez pas l'accessibilité, la méthode est toujours privée.
- ❑ Les exemples de ce cours incluent toujours les mots-clés d'accessibilité, car ils améliorent la clarté.
- ❑ La plupart des programmeurs conviennent que c'est une bonne approche pour explicitement préciser l'accessibilité de votre code.

## 2.14.2. APPEL DES MÉTHODES

## 2.14.2. APPEL DES MÉTHODES

- ❑ L'appel de vos méthodes est simple: vous tapez simplement le nom de la méthode, suivi de parenthèses.
- ❑ Si votre méthode renvoie des données, vous avez la possibilité d'utiliser les données qu'elle renvoie ou simplement de les ignorer :

```
// This call is allowed.  
MyMethodNoReturnedData();
```

```
// This call is allowed.  
MyMethodReturnsData();
```

```
// This call is allowed.  
int myNumber;  
myNumber = MyMethodReturnsData();
```

```
// This call isn't allowed.  
// MyMethodNoReturnedData() does not return any information.  
myNumber = MyMethodNoReturnedData();
```

### 2.14.3. PARAMÈTRES DES MÉTHODES

## 2.14.3. PARAMÈTRES DES MÉTHODES

- ❑ Les méthodes peuvent également accepter des informations via des paramètres.
- ❑ Les paramètres sont déclarés de la même manière que les variables.
- ❑ Par convention, les noms de paramètres commencent toujours par une lettre minuscule dans n'importe quelle langue.

## 2.14.3. PARAMÈTRES DES MÉTHODES

❑ Voici comment créer une fonction qui accepte deux paramètres et renvoie leur somme :

```
private int AddNumbers(int number1, int number2)
{
    return number1 + number2;
}
```

## 2.14.3. PARAMÈTRES DES MÉTHODES

□ Lors de l'appel d'une méthode, vous spécifiez les paramètres requis entre parenthèses ou utilisez un ensemble vide de parenthèses si aucun paramètre n'est requis :



## 2.14.3. PARAMÈTRES DES MÉTHODES

```
// Call a method with no parameters.  
MyMethodNoReturnedData();
```

```
// Call a method that requires two integer parameters.  
MyMethodNoReturnedData2(10, 20);
```

```
// Call a method with two integer parameters and an integer return value.  
int returnValue = AddNumbers(10, 10);
```

## 2.14.4. SURCHARGE DE MÉTHODES

## 2.14.4. SURCHARGE DE MÉTHODES

- ❑ C# prend en charge la surcharge de méthode, ce qui vous permet de créer plusieurs méthodes avec le même nom mais avec un ensemble de paramètres différent.
- ❑ Lorsque vous appelez la méthode, le CLR choisit automatiquement la version correcte en examinant les paramètres que vous fournissez.

## 2.14.4. SURCHARGE DE MÉTHODES

- ❑ Cette technique vous permet de rassembler différentes versions de plusieurs méthodes.
- ❑ Par exemple, vous peut autoriser une recherche dans la base de données qui renvoie un tableau d'objets « Product » représentant des enregistrements dans la base de données.

## 2.14.4. SURCHARGE DE MÉTHODES

- ❑ Plutôt que de créer trois méthodes avec des noms différents selon les critères, tels que `GetAllProducts()`, `GetProductsInCategory()` et `GetActiveProducts()`, vous pouvez créer trois versions de la méthode `GetProducts()`.
- ❑ Chaque méthode aurait le même nom mais une signature différente, ce qui signifie qu'il faudrait spécifier différents paramètres pour chaque méthode.
- ❑ Cet exemple fournit deux versions surchargées pour la méthode `GetProductPrice()` :

```
private decimal GetProductPrice(int ID)
{
    // Code here.
}
```

```
private decimal GetProductPrice(string name)
{
    // Code here.
}
```

```
// And so on...
```

## 2.14.4. SURCHARGE DE MÉTHODES

❑ Vous pouvez désormais rechercher les prix des produits en fonction de l'ID d'un produit unique ou du nom complet du produit, selon si vous fournissez un argument entier ou chaîne :

## 2.14.4. SURCHARGE DE MÉTHODES

```
decimal price;
```

```
// Get price by product ID (the first version).  
price = GetProductPrice(1001);
```

```
// Get price by product name (the second version).  
price = GetProductPrice("DVD Player");
```



## 2.14.4. SURCHARGE DE MÉTHODES

- ❑ Vous ne pouvez pas surcharger une méthode avec des versions qui ont la même signature, c'est-à-dire le même nombre de paramètres et types de données de paramètres, car le CLR ne pourra pas les distinguer les uns des autres.
- ❑ Lorsque vous appelez une méthode surchargée, la version qui correspond à la liste de paramètres que vous fournissez est utilisée.
- ❑ Si aucune version ne correspond, une erreur se produit.

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

- ❑ La surcharge de méthode est une technique traditionnelle pour rendre les méthodes plus flexibles, vous pouvez donc les appeler de diverses façons.
- ❑ C# a également une autre fonctionnalité qui prend en charge le même objectif: les paramètres facultatifs.
- ❑ Un paramètre facultatif est tout paramètre qui a une valeur par défaut.

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

- ❑ Si votre méthode a des paramètres normaux et paramètres facultatifs, les paramètres facultatifs doivent être placés à la fin de la liste des paramètres.
- ❑ Voici un exemple d'une méthode qui a un seul paramètre facultatif:

```
private string GetUserName(int ID, bool useShortForm = false)  
{  
    // Code here.  
}
```

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

❑ Ici, le paramètre `useShortForm` est facultatif, ce qui vous donne deux façons d'appeler la méthode `GetUserName()` :

```
// Explicitly set the useShortForm parameter.  
name = GetUserName(401, true);  
  
// Don't set the useShortForm parameter, and use the default value (false).  
name = GetUserName(401);
```

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

□ Parfois, vous aurez une méthode avec plusieurs paramètres facultatifs, comme celle-ci :

```
private decimal GetSalesTotalForRegion(int regionID, decimal minSale = 0,  
    decimal maxSale = Decimal.MaxValue, bool includeTax = false)  
{  
    // Code here.  
}
```

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

- ❑ Dans cette situation, l'option la plus simple consiste à sélectionner les paramètres que vous souhaitez définir par leur nom.
- ❑ Cette technique est appelée paramètres nommés, et pour l'utiliser, il vous suffit d'ajouter le nom du paramètre suivi de deux points (:), suivi de la valeur, comme indiqué ici :

```
total = GetSalesTotalForRegion(523, maxSale: 5000);
```

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

□ Bien que vous puissiez accomplir plusieurs des mêmes choses avec des paramètres facultatifs et une surcharge de méthode, Les classes sont plus susceptibles d'utiliser la surcharge de méthode pour deux raisons :

- Premièrement, la plupart des classes dans .NET ont été créées dans les versions précédentes, lorsque C# ne prenait pas en charge les paramètres facultatifs.



## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

- Deuxièmement, tous les langages .NET ne prennent pas en charge les paramètres optionnels (bien que C# et VB le fassent).

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

- ❑ Il convient également de noter que la surcharge de méthode vous permet de gérer l'une ou l'autre des situations, bien que les paramètres facultatifs ne le font pas.
- ❑ Par exemple, la méthode `GetProductPrice()` présentée dans la section précédente nécessitait une chaîne ou un entier.

## 2.14.5. PARAMÈTRES OPTIONNELS ET NOMMÉS

- ❑ Il n'est pas acceptable de transformer ces deux paramètres en paramètres facultatifs, car au moins un est requis, et fournir les deux à la fois n'a aucun sens.
- ❑ Il s'agit donc d'une situation où la surcharge de méthode plus naturellement.

## 2.15. LES TABLEAUX

## 2.15. LES TABLEAUX

- ❑ Les tableaux vous permettent de stocker une série de valeurs qui ont le même type de données.
- ❑ Chaque valeur individuelle du tableau est accessible en utilisant un ou plusieurs numéros d'index.
- ❑ Il est souvent pratique d'imaginer les tableaux sous forme de listes de données (si le tableau a une dimension) ou des grilles de données (si le tableau a deux dimensions).

## 2.15. LES TABLEAUX

- ❑ En règle générale, les tableaux sont disposés de manière contiguë en mémoire.
- ❑ Tous les tableaux commencent à une limite inférieure fixe de 0.
- ❑ Cette règle n'a pas d'exceptions.
- ❑ Lorsque vous créez un tableau en C#, vous spécifiez le nombre d'éléments.

## 2.15. LES TABLEAUX

- ❑ Comme le comptage commence à 0, l'indice le plus élevé est en fait un de moins que le nombre d'éléments.
- ❑ En d'autres termes, si vous avez trois éléments, l'indice le plus élevé est 2.

## 2.15. LES TABLEAUX

```
// Create an array with four strings (from index 0 to index 3).  
// You need to initialize the array with the  
// new keyword in order to use it.  
string[] stringArray = new string[4];  
  
// Create a 2x4 grid array (with a total of eight integers).  
int[,] intArray = new int[2, 4];
```



## 2.15. LES TABLEAUX

- ❑ Par défaut, si votre tableau comprend des types de données simples, ils sont tous initialisés aux valeurs par défaut (0 ou False), selon que vous utilisez un type de nombre ou une variable booléenne.
- ❑ Mais si votre tableau se compose de chaînes ou d'un autre type d'objet, il est initialisé avec des références nulles.
- ❑ Pour une discussion plus complète qui décrit la différence entre les types de valeurs simples et les types de référence, le chapitre 3.

## 2.15. LES TABLEAUX

- ❑ Vous pouvez également remplir un tableau avec des données en même temps que vous le créez.
- ❑ Dans ce cas, vous n'avez pas besoin de spécifiez explicitement le nombre d'éléments, car .NET peut le déterminer automatiquement:

```
// Create an array with four strings, one for each number from 1 to 4.  
string[] stringArray = {"1", "2", "3", "4"};
```

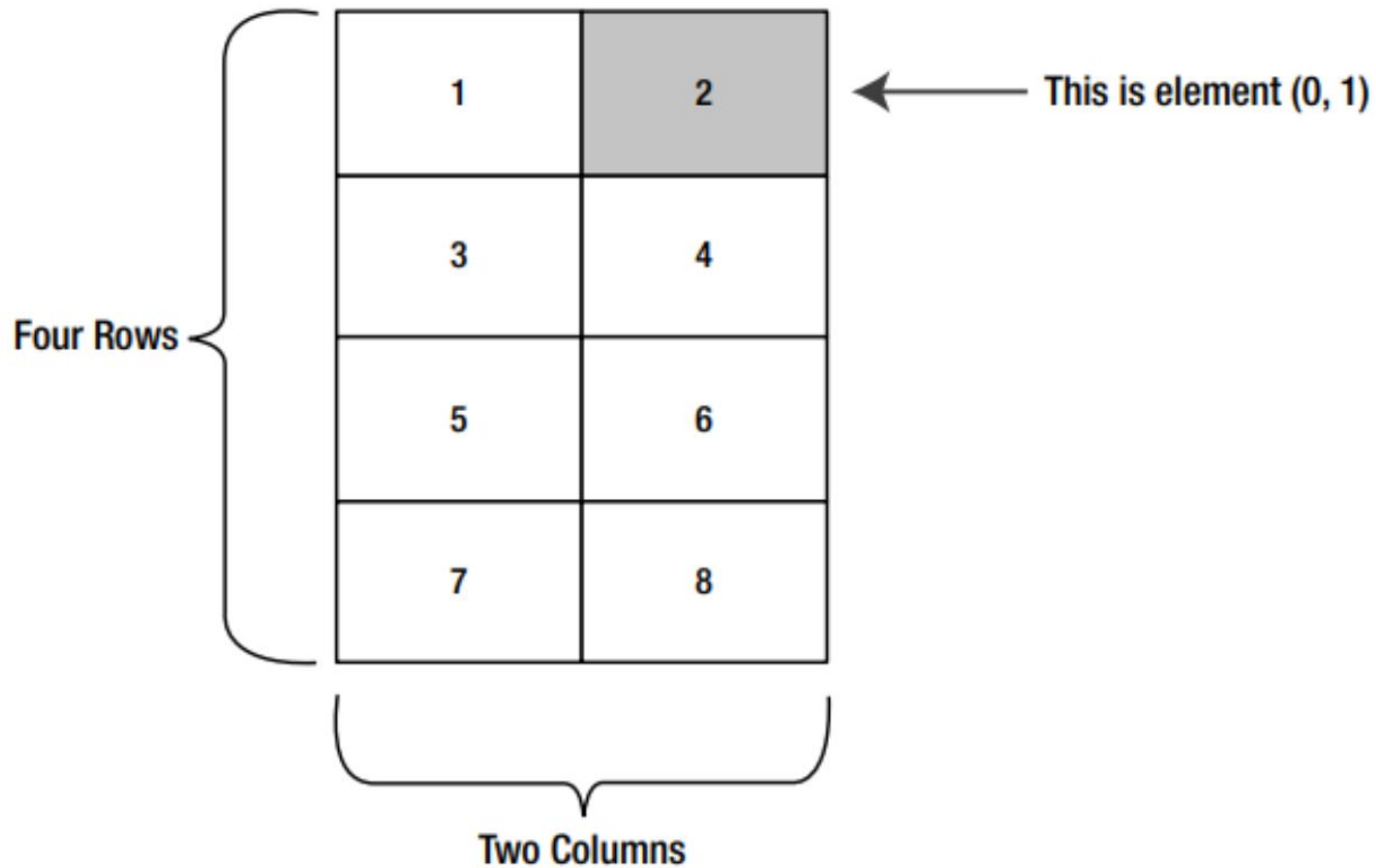
## 2.15. LES TABLEAUX

- ❑ La même technique fonctionne pour les tableaux multidimensionnels, sauf que deux ensembles d'accolades sont nécessaires:

```
// Create a 4x2 array (a grid with four rows and two columns).  
int[,] intArray = {{1, 2}, {3, 4}, {5, 6}, {7, 8}};
```

## 2.15. LES TABLEAUX

□ La figure 2-1 montre à quoi ressemble ce tableau en mémoire.



**Figure 2-1.** *A sample array of integers*

## 2.15. LES TABLEAUX

- ❑ Pour accéder à un élément d'un tableau, vous spécifiez le numéro d'index correspondant entre crochets: [ ].
- ❑ Les indices de tableau sont toujours basés sur zéro.
- ❑ Cela signifie que `myArray[0]` accède à la première cellule d'un tableau à une dimension, `myArray[1]` accède à la deuxième cellule, et ainsi de suite.

```
int[] intArray = {1, 2, 3, 4};  
int element = intArray[2];    // element is now set to 3.
```

## 2.15. LES TABLEAUX

□ Dans un tableau à deux dimensions, vous avez besoin de deux numéros d'index:

```
int[,] intArray = {{1, 2}, {3, 4}, {5, 6}, {7, 8}};
```

```
// Access the value in row 0 (first row), column 1 (second column).  
int element = intArray[0, 1];    // element is now set to 2.
```

## 2.16. LES ArrayList



## 2.16. LES ArrayList

- ❑ Les tableaux C# ne prennent pas en charge le redimensionnement.
- ❑ Cela signifie qu'après avoir créé un tableau, vous ne pouvez pas modifier sa taille.
- ❑ Au lieu de cela, vous devrez créer un nouveau tableau avec la nouvelle taille et copier les valeurs de l'ancien tableau vers le nouveau, ce qui serait un processus fastidieux.

## 2.16. LES ArrayList

- ❑ Cependant, si vous avez besoin d'une liste de type tableau dynamique, vous pouvez utiliser l'un des classes de collection fournies à tous les langages .NET via la bibliothèque de classes .NET.
- ❑ Une des collections les plus simples que les classes .NET proposent est le ArrayList, qui prend en charge tout type d'objet et permet toujours le redimensionnement dynamique.
- ❑ Voici un extrait de code C# qui utilise un ArrayList:

```
// Create an ArrayList object. It's a collection, not an array,  
// so the syntax is slightly different.
```

```
ArrayList dynamicList = new ArrayList();
```

```
// Add several strings to the list.
```

```
// The ArrayList is not strongly typed, so you can add any data type  
// although it's simplest if you store just one type of object  
// in any given collection.
```

```
dynamicList.Add("one");
```

```
dynamicList.Add("two");
```

```
dynamicList.Add("three");
```

```
// Retrieve the first string. Notice that the object must be converted to a  
// string, because there's no way for .NET to be certain what it is.
```

```
string item = Convert.ToString(dynamicList[0]);
```

## 2.16. LES ArrayList

❑ Vous en apprendrez plus sur les ArrayList et d'autres collections au chapitre 3.