**Module 1: Examen de la syntaxe Visual C #**

**Contenu:**

|  |  |
| --- | --- |
|  | [Présentation du module](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C01.html#P1) |
| **Leçon 1:** | [Vue d'ensemble de l'écriture d'application à l'aide de Visual C #](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C01.html#P2) |
| **Leçon 2:** | [Types de données, opérateurs et expressions](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C01.html#P3) |
| **Lecon 3:** | [Constructions du langage de programmation Visual C #](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C01.html#P4) |
| **Laboratoire:** | [Développement de l'application d'inscription aux classes](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C01.html#P5) |
|  | [Revue du module et points à retenir](https://www.skillpipe.com/api/2.1/content/urn:uuid:21ae3267-8517-4025-bf9a-2dd83635ed12@2020-12-11T20:16:12Z/OPS/html/20483C01.html#P6) |

***Table des matières***

[Contenu: 1](#_Toc66906620)

[Table des matières 2](#_Toc66906621)

[I. Présentation du module 6](#_Toc66906622)

[A. Objectifs 6](#_Toc66906623)

[II. Leçon 1: 7](#_Toc66906624)

[Vue d'ensemble de l'écriture d'application à l'aide de Visual C # 7](#_Toc66906625)

[A. Objectifs de la leçon 7](#_Toc66906626)

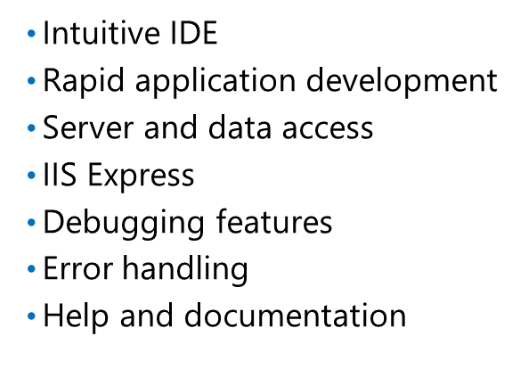
[B. Qu'est-ce que le .NET Framework? 8](#_Toc66906627)

[1. Le Common Language Runtime 8](#_Toc66906628)

[2. La bibliothèque de classes .NET Framework 9](#_Toc66906629)

[3. Cadres de développement 9](#_Toc66906630)

[C. Principales fonctionnalités de Visual Studio 2017 10](#_Toc66906631)

[ 10](#_Toc66906632)

[D. Modèles dans Visual Studio 2017 11](#_Toc66906633)

[1. Types de modèles 12](#_Toc66906634)

[E. Création d'une application .NET Framework 13](#_Toc66906635)

[1. Fonctionnalités de productivité du programmeur 15](#_Toc66906636)

[F. Vue d'ensemble de XAML 16](#_Toc66906637)

[1. Définition de contrôles en XAML 17](#_Toc66906638)

[III. Leçon 2: Types de données, opérateurs et expressions 18](#_Toc66906639)

[A. Objectifs de la leçon 18](#_Toc66906640)

[B. Que sont les types de données? 18](#_Toc66906641)

[1. Types de données couramment utilisés 19](#_Toc66906642)

[C. Expressions et opérateurs dans Visual C # 20](#_Toc66906643)

[1. Opérateurs dans Visual C # 22](#_Toc66906644)

[2. Exemples d'expressions 23](#_Toc66906645)

[a) + Opérateur 23](#_Toc66906646)

[b) / Opérateur 24](#_Toc66906647)

[c) / Opérateur 24](#_Toc66906648)

[d) + et – Opérateurs 24](#_Toc66906649)

[e) + Opérateur 25](#_Toc66906650)

[D. Déclaration et attribution de variables 25](#_Toc66906651)

[1. Déclarer et attribuer une variable 26](#_Toc66906652)

[a) Déclaration d'une variable 27](#_Toc66906653)

[b) Assigner une variable 27](#_Toc66906654)

[c) Déclaration et attribution de variables 28](#_Toc66906655)

[2. Variables implicitement typées 28](#_Toc66906656)

[a) Déclaration d'une variable à l'aide du mot-clé var 28](#_Toc66906657)

[3. Variables d'objet 28](#_Toc66906658)

[a) Le nouvel opérateur 29](#_Toc66906659)

[E. Accès aux membres de type 29](#_Toc66906660)

[1. Invoquer des membres 30](#_Toc66906661)

[F. Diffusion entre les types de données 31](#_Toc66906662)

[1. Conversions implicites 32](#_Toc66906663)

[a) Conversion implicite 32](#_Toc66906664)

[2. Conversions explicites 33](#_Toc66906665)

[a) Conversion explicite 33](#_Toc66906666)

[3. Utilisation de la classe System.Convert 34](#_Toc66906667)

[a) Conversions à l'aide de la méthode ToInt32 34](#_Toc66906668)

[b) TryParse Conversion 34](#_Toc66906669)

[G. Manipuler des chaînes 35](#_Toc66906670)

[1. Concaténation de chaînes 36](#_Toc66906671)

[a) Concaténation à l'aide de l'opérateur + 37](#_Toc66906672)

[b) Concaténation à l'aide de la classe StringBuilder 37](#_Toc66906673)

[2. Validation des chaînes 37](#_Toc66906674)

[a) Regex.IsMatch, méthode 38](#_Toc66906675)

[IV. Leçon 3: Constructions du langage de programmation Visual C # 38](#_Toc66906676)

[A. Objectifs de la leçon 39](#_Toc66906677)

[B. Implémentation de la logique conditionnelle 39](#_Toc66906678)

[1. Instructions conditionnelles 40](#_Toc66906679)

[a) Instruction if 41](#_Toc66906680)

[b) Instruction if else 41](#_Toc66906681)

[c) Instruction else if 42](#_Toc66906682)

[d) 43](#_Toc66906683)

[2. Instructions de sélection 43](#_Toc66906684)

[a) Instruction switch 43](#_Toc66906685)

[C. Implémentation de la logique d'itération 45](#_Toc66906686)

[a) Boucle For 46](#_Toc66906687)

[b) boucle foreach 47](#_Toc66906688)

[c) Boucle while 47](#_Toc66906689)

[d) Boucle do 48](#_Toc66906690)

[D. Création et utilisation de tableaux 49](#_Toc66906691)

[1. Création de tableaux 50](#_Toc66906692)

[a) Accès aux données par index 51](#_Toc66906693)

[b) Itération sur un tableau 52](#_Toc66906694)

[E. Référencer les espaces de noms 53](#_Toc66906695)

[1. Espaces de noms de la bibliothèque de classes .NET Framework 53](#_Toc66906696)

[2. Espaces de noms définis par l'utilisateur 54](#_Toc66906697)

[a) Définition d'un espace de noms 54](#_Toc66906698)

[3. Utilisation des espaces de noms 55](#_Toc66906699)

[a) Importer un espace de noms 55](#_Toc66906700)

[b) Appeler la console par son nom complet 56](#_Toc66906701)

[F. Utilisation de points d'arrêt dans Visual Studio 2017 56](#_Toc66906702)

[1. Utilisation des points d'arrêt 56](#_Toc66906703)

[2. Utilisation des contrôles de débogage 57](#_Toc66906704)

[V. Démonstration: développement du laboratoire d'application d'inscription aux classes 58](#_Toc66906705)

[**Démonstration étapes** 58](#_Toc66906706)

[A. Laboratoire: Développer l'application d'inscription aux classes 59](#_Toc66906707)

[**Scénario** 59](#_Toc66906708)

[B. Objectifs 59](#_Toc66906709)

[1. Configuration du laboratoire 60](#_Toc66906710)

[2. Exercice 1: Implémentation de la fonctionnalité d'édition pour la liste des étudiants 60](#_Toc66906711)

[a) Scénario 60](#_Toc66906712)

[3. Exercice 2: Implémentation de la fonctionnalité d'insertion pour la liste des étudiants 61](#_Toc66906713)

[C. Scénario 61](#_Toc66906714)

[1. Exercice 3: Implémentation de la fonctionnalité de suppression pour la liste des étudiants 61](#_Toc66906715)

[VI. Scénario 61](#_Toc66906716)

[1. Exercice 4: Affichage de l'âge d'un élève 62](#_Toc66906717)

[a) Scénario 62](#_Toc66906718)

[Revue du module et points à retenir 62](#_Toc66906719)

[**Question (s) de révision** 62](#_Toc66906720)

# Présentation du module

Microsoft® .NET Framework version 4.7 fournit une plate-forme de développement complète que vous pouvez utiliser pour créer, déployer et gérer des applications et des services. En utilisant le .NET Framework, vous pouvez créer des applications visuellement attrayantes, permettre une communication transparente au-delà des frontières technologiques et prendre en charge un large éventail de processus métier.

Dans ce module, vous découvrirez certaines des fonctionnalités de base fournies par .NET Framework et Microsoft Visual Studio®. Vous découvrirez également certaines des constructions de base de Visual C # ® qui vous permettent de commencer à développer des applications .NET Framework.

## Objectifs

Après avoir terminé ce module, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Décrivez l'architecture des applications .NET Framework et les fonctionnalités fournies par Visual Studio 2017 et Visual C #. |
| • | Utilisez des types de données, des opérateurs et des expressions Visual C # de base. |
| • | Utilisez des constructions Visual C # standard. |

# Leçon 1:

# Vue d'ensemble de l'écriture d'application à l'aide de Visual C #

Le .NET Framework 4.7 et Visual Studio fournissent de nombreuses fonctionnalités que vous pouvez utiliser lors du développement de vos applications.

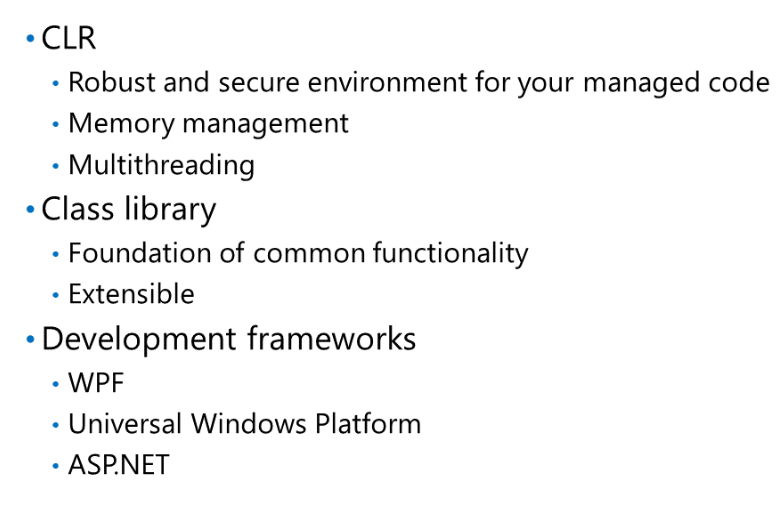
Dans cette leçon, vous découvrirez les fonctionnalités fournies par Visual Studio 2017 et .NET Framework 4.7 qui vous permettent de créer vos propres applications.

## Objectifs de la leçon

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Décrivez l'objectif du .NET Framework. |
| • | Décrivez les principales fonctionnalités de Visual Studio 2017. |
| • | Décrivez les modèles de projet fournis dans Visual Studio 2017. |
| • | Créez une application .NET Framework. |
| • | Décrivez XAML. |

## Qu'est-ce que le .NET Framework?



Le .NET Framework 4.7 fournit une plate-forme de développement complète qui offre un moyen rapide et efficace de créer des applications et des services. En utilisant Visual Studio 2017, vous pouvez utiliser le .NET Framework 4.7 pour créer une large gamme de solutions qui fonctionnent sur une large gamme de périphériques informatiques.

Le .NET Framework 4.7 fournit trois éléments principaux:

|  |  |
| --- | --- |
| • | Le Common Language Runtime (CLR). |
| • | La bibliothèque de classes .NET Framework. |
| • | Une collection de cadres de développement. |

### Le Common Language Runtime

Le .NET Framework fournit un environnement appelé CLR. Le CLR gère l'exécution du code et simplifie le processus de développement en fournissant un environnement d'exécution robuste et hautement sécurisé qui comprend:

|  |  |
| --- | --- |
| • | Gestion de la mémoire. |
| • | Transactions. |
| • | Multithreading. |

### La bibliothèque de classes .NET Framework

Le .NET Framework fournit une bibliothèque de classes réutilisables que vous pouvez utiliser pour créer des applications. Les classes fournissent une base de fonctionnalités et de constructions communes qui aident à simplifier le développement d'applications en éliminant en partie le besoin de réinventer constamment la logique. Par exemple, la classe System.IO.File contient des fonctionnalités qui vous permettent de manipuler des fichiers sur le système de fichiers Windows. En plus d'utiliser les classes de la bibliothèque de classes .NET Framework, vous pouvez étendre ces classes en créant vos propres bibliothèques de classes.

### Cadres de développement

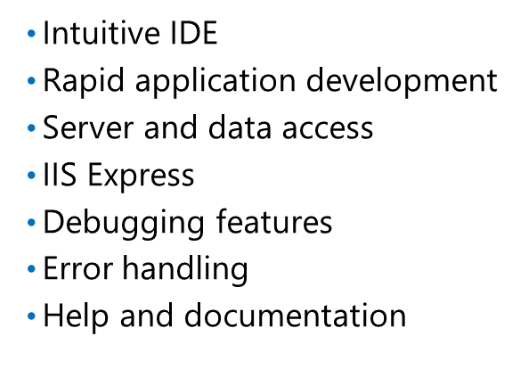
Le .NET Framework fournit plusieurs frameworks de développement que vous pouvez utiliser pour créer des types d'applications courants, notamment:

|  |  |
| --- | --- |
| • | Applications client de bureau, à l'aide de Windows Presentation Foundation (WPF). |
| • | Applications de la plateforme Windows universelle (UWP), à l'aide de XAML. |
| • | Applications Web côté serveur, à l'aide des applications Web Microsoft ASP.NET ou ASP.NET MVC. |
| • | Applications Web orientées services, à l'aide de l'API Web ASP.NET. |
| • | Applications de longue durée, en utilisant les services Windows. |

Chaque cadre fournit les composants et l'infrastructure nécessaires pour vous aider à démarrer.

**Lecture supplémentaire:**Pour plus d'informations sur le .NET Framework, consultez la page Vue d'ensemble de .NET Framework à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267639>.

## Principales fonctionnalités de Visual Studio 2017



Visual Studio 2017 fournit un environnement de développement unique qui vous permet de concevoir, implémenter, créer, tester et déployer rapidement divers types d'applications et de composants à l'aide d'une gamme de langages de programmation.

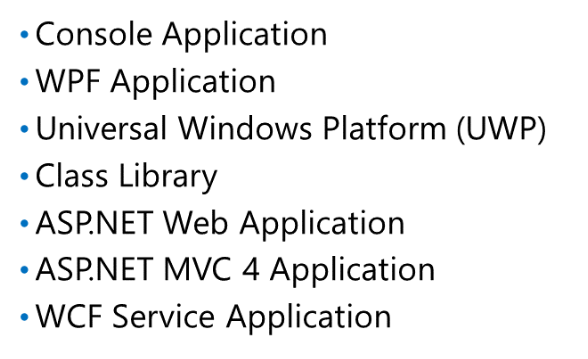
Certaines des fonctionnalités clés de Visual Studio 2017 sont:

|  |  |
| --- | --- |
| • | Environnement de développement intégré intuitif (IDE). L'IDE de Visual Studio 2017 fournit toutes les fonctionnalités et tous les outils nécessaires pour concevoir, implémenter, créer, tester et déployer des applications et des composants. |
| • | Développement rapide d'applications. Visual Studio 2017 fournit des vues de conception pour les composants graphiques qui vous permettent de créer facilement des interfaces utilisateur complexes. Vous pouvez également utiliser les vues de l'éditeur de code, qui offrent plus de contrôle mais ne sont pas aussi faciles à utiliser. Visual Studio 2017 fournit également des assistants qui aident à accélérer le développement de composants particuliers. |
| • | Accès au serveur et aux données. Visual Studio 2017 fournit l'Explorateur de serveurs, qui vous permet de vous connecter aux serveurs et d'explorer leurs bases de données et services système. Il fournit également un moyen familier de créer, d'accéder et de modifier les bases de données utilisées par votre application à l'aide du nouveau concepteur de tables. |
| • | Services Internet (IIS) Express. Visual Studio 2017 fournit une version allégée d'IIS en tant que serveur Web par défaut pour le débogage de vos applications Web. |
| • | Fonctionnalités de débogage. Visual Studio 2017 fournit un débogueur qui vous permet de parcourir le code local ou distant, de faire une pause aux points d'arrêt et de suivre les chemins d'exécution. |
| • | La gestion des erreurs. Visual Studio 2017 fournit la fenêtre Liste des erreurs, qui affiche les erreurs, avertissements ou messages générés lors de la modification et de la génération de votre code. |
| • | Aide et documentation. Visual Studio 2017 fournit une aide et des conseils via Microsoft IntelliSense®, des extraits de code et le système d'aide intégré, qui contient de la documentation et des exemples. |

**Lecture supplémentaire:**Pour plus d'informations sur les nouveautés de Visual Studio 2017, consultez la page Nouveautés de Visual Studio 2017 à l'adresse

<https://aka.ms/moc-20483c-m1-pg1>

## Modèles dans Visual Studio 2017



Visual Studio 2017 prend en charge le développement de différents types d'applications telles que les applications clientes Windows, les applications Web, les services et les bibliothèques. Pour vous aider à démarrer, Visual Studio 2017 fournit des modèles d'application qui fournissent une structure pour les différents types d'applications. Ces modèles:

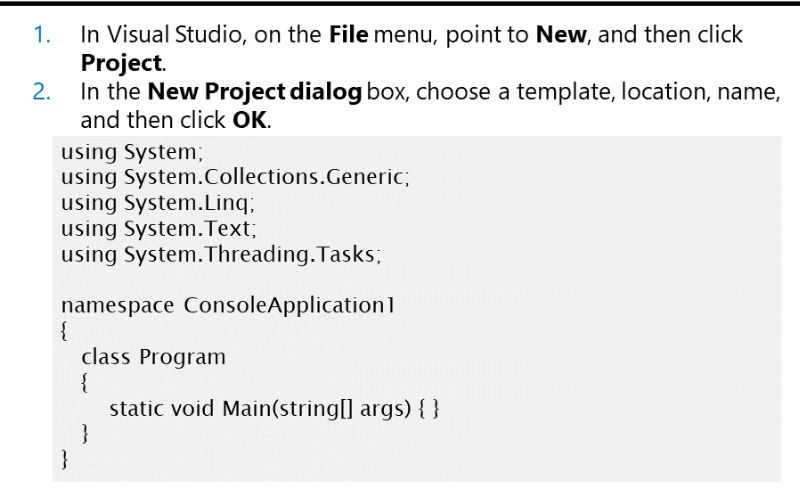
|  |  |
| --- | --- |
| • | Fournissez un code de démarrage sur lequel vous pouvez construire pour créer rapidement des applications fonctionnelles. |
| • | Incluez les composants de support et les contrôles qui sont pertinents pour le type de projet. |
| • | Configurez l'IDE de Visual Studio 2017 sur le type d'application que vous développez. |
| • | Ajoutez des références à tous les assemblys initiaux requis par ce type d'application. |

### Types de modèles

Le tableau suivant décrit certains des modèles d'application courants que vous pouvez utiliser lorsque vous développez des applications .NET Framework à l'aide de Visual Studio 2017.

| **Modèle** | **Description** |
| --- | --- |
| Application de la console | Fournit les paramètres d'environnement, les outils, les références de projet et le code de démarrage pour développer une application qui s'exécute dans une interface de ligne de commande. Ce type d'application est considéré comme léger car il n'y a pas d'interface utilisateur graphique. |
| Application WPF | Fournit les paramètres d'environnement, les outils, les références de projet et le code de démarrage pour créer une application Windows graphique riche. Une application WPF vous permet de créer la prochaine génération d'applications Windows, avec beaucoup plus de contrôle sur la conception de l'interface utilisateur. |
| UWP | Fournit les paramètres d'environnement, les outils, les références de projet et le code de démarrage pour créer une application graphique riche ciblée sur les systèmes d'exploitation Windows 10. Les applications UWP vous permettent de réutiliser les compétences obtenues du développement WPF en utilisant XAML et Visual C #, mais également du développement Web en utilisant HTML 5, CSS 3.0 et JavaScript. |
| Bibliothèque de classe | Fournit les paramètres d'environnement, les outils et le code de démarrage pour créer un assembly .dll. Vous pouvez utiliser ce type de fichier pour stocker des fonctionnalités que vous souhaiterez peut-être appeler à partir de nombreuses autres applications. |
| Application Web ASP.NET | Fournit les paramètres d'environnement, les outils, les références de projet et le code de démarrage pour créer une application Web ASP.NET compilée côté serveur. |
| Application ASP.NET MVC 4 | Fournit les paramètres d'environnement, les outils, les références de projet et le code de démarrage pour créer une application Web Model-View-Controller (MVC). Une application Web ASP.NET MVC diffère de l'application Web ASP.NET standard en ce que l'architecture de l'application vous aide à séparer la couche de présentation, la couche de logique métier et la couche d'accès aux données. |
| Application de service WCF | Fournit les paramètres d'environnement, les outils, les références de projet et le code de démarrage pour créer des services SOA (Service Orientated Architecture). |

## Création d'une application .NET Framework



Les modèles d'application fournis dans Visual Studio 2017 vous permettent de commencer à créer une application avec un minimum d'effort. Vous pouvez ensuite ajouter votre code et personnaliser le projet pour répondre à vos propres besoins.

Les étapes suivantes décrivent comment créer une application console:

|  |  |
| --- | --- |
| 1. | Ouvrez Visual Studio 2017. |
| 2. | Dans Visual Studio, dans le menu Fichier, pointez sur Nouveau, puis cliquez sur Projet. |
| 3. | Dans la boîte de dialogue Nouveau projet, procédez comme suit:   |  |  | | --- | --- | | une. | Développez Modèles, Visual C #, puis cliquez sur Windows. | | b. | Cliquez sur le modèle d'application console. | | c. | Dans la zone Nom, spécifiez un nom pour le projet. | | ré. | Dans la zone Emplacement, spécifiez le chemin où vous souhaitez enregistrer le projet. | |
| 4. | Cliquez sur OK. |
| 5. | La fenêtre de l'éditeur de code affiche maintenant la classe Program par défaut, qui contient la méthode de point d'entrée de l'application. |

L'exemple de code suivant montre la classe Program par défaut fournie par Visual Studio lorsque vous utilisez le modèle d'application console.

***Program Class***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

}

}

}

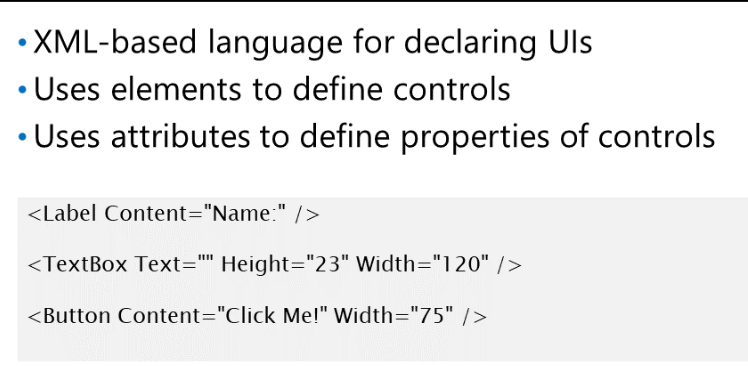
Après avoir créé un projet, vous pouvez utiliser les fonctionnalités fournies par Visual Studio pour créer votre application.

### Fonctionnalités de productivité du programmeur

Visual Studio 2017 fournit une foule de fonctionnalités qui peuvent vous aider à écrire du code. Lors de l'écriture de code, vous devez rappeler des informations sur de nombreux éléments du programme. Au lieu de rechercher manuellement des informations en recherchant des fichiers d'aide ou un autre code source, la fonctionnalité IntelliSense de Visual Studio fournit les informations dont vous avez besoin directement à partir de l'éditeur. IntelliSense fournit les fonctionnalités suivantes:

|  |  |
| --- | --- |
| • | L'option Quick Info affiche la déclaration complète de tout identifiant de votre code. Déplacez la souris de sorte que le pointeur repose sur un identificateur pour afficher les informations rapides pour cet identificateur, qui apparaît dans une boîte contextuelle jaune. |
| • | L'option Compléter le mot entre le reste d'une variable, une commande ou un nom de fonction après avoir tapé suffisamment de caractères pour lever l'ambiguïté du terme. Tapez les premières lettres du nom, puis appuyez sur Alt + Flèche droite ou Ctrl + Barre d'espace pour terminer le mot. |

## Vue d'ensemble de XAML



Le langage XAML (Extensible Application Markup Language) est un langage XML que vous pouvez utiliser pour définir les interfaces utilisateur de vos applications .NET. En déclarant votre interface utilisateur en XAML au lieu de l'écrire dans du code, votre interface utilisateur est plus portable et sépare votre interface utilisateur de la logique de votre application.

XAML utilise des éléments et des attributs pour définir des contrôles et leurs propriétés dans la syntaxe XML. Lorsque vous concevez une interface utilisateur, vous pouvez utiliser la boîte à outils et le volet des propriétés dans Visual Studio pour créer visuellement l'interface utilisateur, vous pouvez utiliser le volet XAML pour créer de manière déclarative l'interface utilisateur, vous pouvez utiliser Microsoft Expression Blend ou vous pouvez utiliser un autre tiers outils. L'utilisation du volet XAML vous donne un contrôle plus fin que le fait de faire glisser des contrôles de la boîte à outils vers la fenêtre.

L'exemple suivant montre la déclaration XAML pour une étiquette, une zone de texte et un bouton:

### Définition de contrôles en XAML

<Label Content="Name:"/>

<TextBox Text="" Height="23" Width="120" />

<Button Content="Click Me!" Width="75" />

Vous pouvez utiliser la syntaxe XAML pour produire des interfaces utilisateur simples comme indiqué dans l'exemple précédent ou pour créer des interfaces beaucoup plus complexes. La syntaxe de balisage fournit la fonctionnalité permettant de lier des données à des contrôles, d'utiliser des dégradés et des textures, d'utiliser des modèles pour la mise en forme à l'échelle de l'application et de lier des événements à des contrôles dans la fenêtre. La boîte à outils de Visual Studio comprend également des contrôles de conteneur que vous pouvez utiliser pour positionner et dimensionner vos contrôles de manière appropriée, quelle que soit la façon dont vos utilisateurs redimensionnent la fenêtre de leur application.

**Lecture supplémentaire:**Pour plus d'informations sur XAML, consultez le module 9 de ce cours.

# Leçon 2: Types de données, opérateurs et expressions

Toutes les applications utilisent des données. Ces données peuvent être fournies par l'utilisateur via une interface utilisateur, à partir d'une base de données, d'un service réseau ou d'une autre source. Pour stocker et utiliser des données dans vos applications, vous devez vous familiariser avec la façon de définir et d'utiliser des variables et de créer et d'utiliser des expressions avec la variété d'opérateurs fournis par Visual C #.

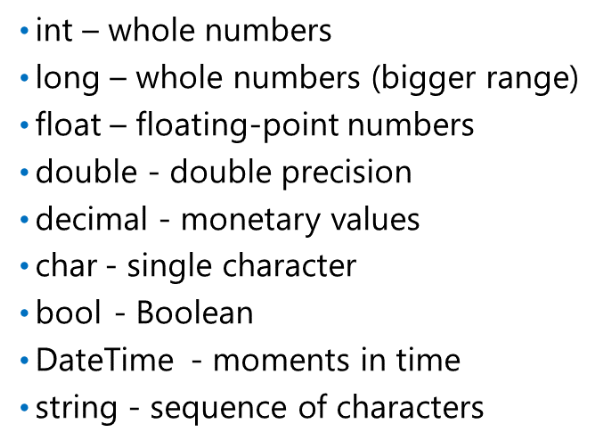
Dans cette leçon, vous apprendrez à utiliser certaines des constructions fondamentales de Visual C #, telles que les variables, les membres de type, la conversion et la manipulation de chaînes.

## Objectifs de la leçon

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Décrivez les types de données fournis par Visual C #. |
| • | Créez et utilisez des expressions. |
| • | Déclarez et attribuez des variables. |
| • | Accéder aux membres du type. |
| • | Diffusez des données d'un type à un autre. |
| • | Concaténez et validez les chaînes |

## Que sont les types de données?



Une variable contient des données d'un type spécifique. Lorsque vous déclarez une variable pour stocker des données dans une application, vous devez choisir un type de données approprié pour ces données. Visual C # est un langage de type sécurisé, ce qui signifie que le compilateur garantit que les valeurs stockées dans les variables sont toujours du type approprié.

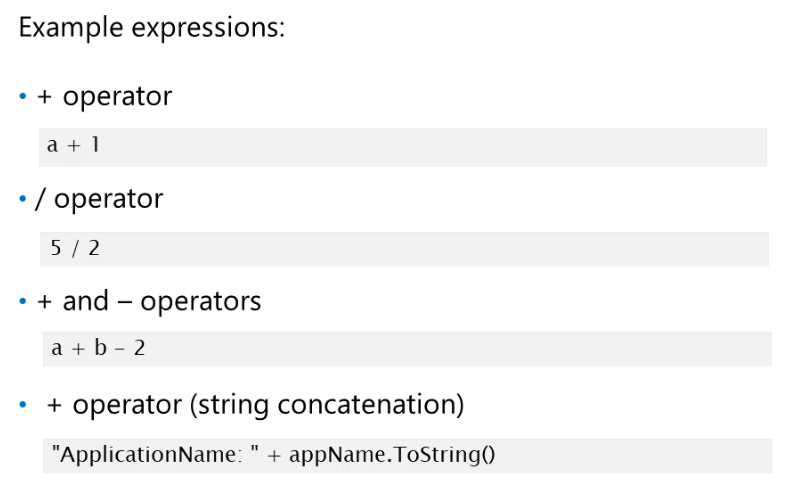
### Types de données couramment utilisés

Le tableau suivant présente les types de données couramment utilisés dans Visual C # et leurs caractéristiques.

| **Type** | **Description** | **Size (bytes)** | **Range** |
| --- | --- | --- | --- |
| int | Whole numbers | 4 | –2,147,483,648 to 2,147,483,647 |
| long | Whole numbers (bigger range) | 8 | –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | Floating-point numbers | 4 | +/–3.4 × 10^38 |
| double | Double precision (more accurate) floating-point numbers | 8 | +/–1.7 × 10^308 |
| decimal | Monetary values | 16 | 28 significant figures |
| char | Single character | 2 | N/A |
| bool | Boolean | 1 | True or false |
| DateTime | Moments in time | 8 | 0:00:00 on 01/01/2001 to 23:59:59 on 12/31/9999 |
| string | Sequence of characters | 2 per character | N/A |

**Lecture supplémentaire:**Pour plus d'informations sur les types de données, consultez la page Tables de référence pour les types (Référence C #) à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267770>.

## Expressions et opérateurs dans Visual C #



Les expressions sont un composant central de pratiquement toutes les applications Visual C #, car les expressions sont les constructions fondamentales que vous utilisez pour évaluer et manipuler les données. Les expressions sont des collections d'opérandes et d'opérateurs, que vous pouvez définir comme suit:

|  |  |
| --- | --- |
| • | Les opérandes sont des valeurs, par exemple des nombres et des chaînes. Il peut s'agir de valeurs constantes (littérales), de variables, de propriétés ou de valeurs renvoyées par des appels de méthode. |
| • | Les opérateurs définissent les opérations à effectuer sur les opérandes, par exemple, l'addition ou la multiplication. Des opérateurs existent pour toutes les opérations mathématiques de base, ainsi que pour des opérations plus avancées telles que la comparaison logique ou la manipulation des bits de données qui constituent une valeur. |

Toutes les expressions sont évaluées à une valeur unique lorsque votre application s'exécute. Le type de valeur qu'une expression produit dépend des types d'opérandes que vous utilisez et des opérateurs que vous utilisez. Il n'y a pas de limite à la longueur des expressions dans les applications Visual C #, bien qu'en pratique, vous soyez limité par la mémoire de votre ordinateur et votre patience lors de la frappe. Cependant, il est généralement conseillé d'utiliser des expressions plus courtes et d'assembler les résultats du traitement d'expression au coup par coup. Cela vous permet de voir plus facilement ce que fait votre code et facilite le débogage de votre code.

### Opérateurs dans Visual C #

Les opérateurs combinent les opérandes en expressions. Visual C # fournit un large éventail d'opérateurs que vous pouvez utiliser pour effectuer les opérations mathématiques et logiques les plus fondamentales. Les opérateurs appartiennent aux trois catégories suivantes:

|  |  |
| --- | --- |
| • | Unaire. Ce type d'opérateur opère sur un seul opérande. Par exemple, vous pouvez utiliser l'opérateur - comme opérateur unaire. Pour ce faire, vous le placez immédiatement avant un opérande numérique et il convertit la valeur de l'opérande en sa valeur actuelle multipliée par –1. |
| • | Binaire. Ce type d'opérande fonctionne sur deux valeurs. C'est le type d'opérateur le plus courant, par exemple, \*, qui multiplie la valeur de deux opérandes. |
| • | Ternaire. Il n'y a qu'un seul opérateur ternaire dans Visual C #. C'est le ? : opérateur utilisé dans les expressions conditionnelles. |

Le tableau suivant montre les opérateurs que vous pouvez utiliser dans Visual C #, regroupés par type.

| **Type** | **Operators** |
| --- | --- |
| Order of operations | ( ) |
| Arithmetic | +, -, \*, /, % |
| Increment, decrement | ++, -- |
| Comparison | ==, !=, <, >, <=, >=, is, ?? |
| String concatenation | + |
| Logical/bitwise operations | &, |, ^, !, ~, &&, || |
| Indexing (counting starts from element 0) | [ ] |
| Casting | ( ), as |
| Assignment | =, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>= |
| Bit shift | <<, >> |
| Type information | sizeof, typeof |
| Delegate concatenation and removal | +, - |
| Overflow exception control | checked, unchecked |
| Indirection and Address (unsafe code only) | \*, ->, [ ], & |
| Conditional (ternary operator) | ?: |

### Exemples d'expressions

Vous pouvez combiner les éléments de base des opérateurs et des opérandes pour créer des expressions aussi simples ou aussi complexes que vous le souhaitez.

L'exemple de code suivant montre comment utiliser l'opérateur +.

#### + Opérateur

a + 1

L'opérateur + peut opérer sur différents types de données et le résultat de cette expression dépend des types de données des opérandes. Par exemple, si a est un entier, le résultat de l'expression est un entier dont la valeur 1 est supérieure à a. Si a est un double, le résultat est un double avec la valeur 1 supérieure à a. La différence est subtile mais importante. Dans le second cas (a est un double), le compilateur Visual C # doit générer du code pour convertir la valeur entière constante 1 en la valeur double constante 1 avant que l'expression puisse être évaluée. La règle est que le type de l'expression est le même que le type des opérandes, bien qu'un ou plusieurs des opérandes puissent devoir être convertis pour s'assurer qu'ils sont tous compatibles.

L'exemple de code suivant montre comment utiliser l'opérateur / pour diviser deux valeurs int.

#### / Opérateur

5/2

La valeur du résultat est la valeur entière 2 (et non 2,5). Si vous convertissez l'un des opérandes en double, le compilateur Visual C # convertira l'autre opérande en double et le résultat sera un double.

L'exemple de code suivant montre comment utiliser l'opérateur / pour diviser une valeur double par une valeur int.

#### / Opérateur

5,0 / 2

La valeur du résultat est maintenant la valeur double 2,5. Vous pouvez continuer à créer des expressions avec des valeurs et des opérateurs supplémentaires.

L'exemple de code suivant montre comment utiliser les opérateurs + et - dans une expression.

#### + et – Opérateurs

a + b - 2

Cette expression évalue la somme des variables a et b avec la valeur 2 soustraite du résultat.

Certains opérateurs, tels que +, peuvent être utilisés pour évaluer des expressions qui ont une plage de types.

L'exemple de code suivant montre comment utiliser l'opérateur + pour concaténer deux valeurs de chaîne.

#### + Opérateur

"ApplicationName:" + appName.ToString ()

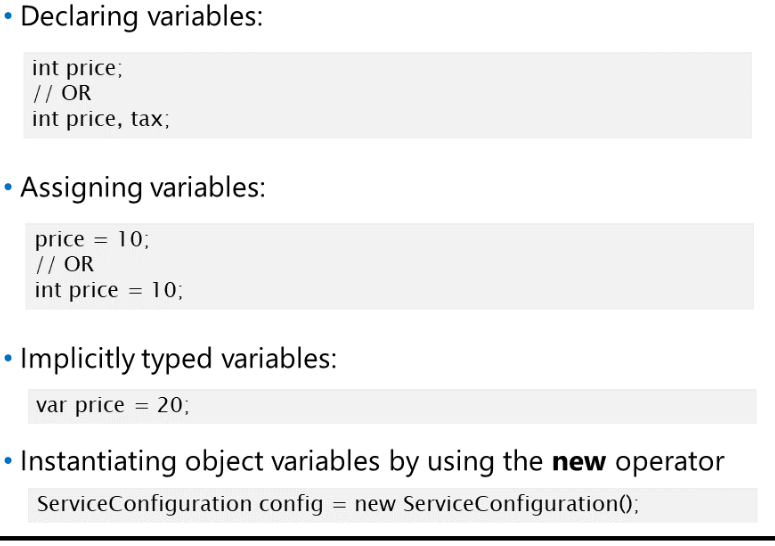
L'opérateur + utilise un opérande qui est le résultat d'un appel de méthode, ToString (). La méthode ToString () convertit la valeur d'une variable en une chaîne, quel que soit son type.

La bibliothèque de classes .NET Framework contient de nombreuses méthodes supplémentaires que vous pouvez utiliser pour effectuer des opérations mathématiques et de chaîne sur des données, telles que la classe System.Math.

**Lecture supplémentaire:**Pour plus d'informations sur les opérateurs, consultez la page Opérateurs C # à l'adresse

<https://aka.ms/moc-20483c-m1-pg2>.

## Déclaration et attribution de variables



Avant de pouvoir utiliser une variable, vous devez la déclarer afin de pouvoir spécifier son nom et ses caractéristiques. Le nom d'une variable est appelé identificateur. Visual C # a des règles spécifiques concernant les identificateurs que vous pouvez utiliser:

|  |  |
| --- | --- |
| • | Un identificateur ne peut contenir que des lettres, des chiffres et des caractères de soulignement. |
| • | Un identifiant doit commencer par une lettre ou un trait de soulignement. |
| • | Un identificateur pour une variable ne doit pas être l'un des mots-clés que Visual C # réserve pour son propre usage. |

Visual C # est sensible à la casse. Si vous utilisez le nom MyData comme identificateur d'une variable, ce n'est pas la même chose que myData. Vous pouvez déclarer deux variables en même temps appelées MyData et myData et Visual C # ne les confondra pas, bien que ce ne soit pas une bonne pratique de codage.

Lorsque vous déclarez des variables, vous devez utiliser des noms significatifs pour vos variables, car cela peut faciliter la compréhension de votre code. Vous devriez également adopter une convention de dénomination et l'utiliser!

### Déclarer et attribuer une variable

Lorsque vous déclarez une variable, vous réservez de l'espace de stockage pour cette variable en mémoire et le type de données qu'elle contiendra. Vous pouvez déclarer plusieurs variables dans une seule déclaration en utilisant le séparateur virgule; toutes les variables déclarées de cette manière ont le même type.

L'exemple suivant montre comment déclarer une nouvelle variable.

#### Déclaration d'une variable

// DataType variableName;

int price;

// OR

// DataType variableName1, variableName2; int price, tax;

Après avoir déclaré une variable, vous pouvez lui attribuer une valeur à l'aide d'une instruction d'affectation. Vous pouvez modifier la valeur d'une variable autant de fois que vous le souhaitez pendant l'exécution de l'application. L'opérateur d'affectation = affecte une valeur à une variable.

L'exemple de code suivant montre comment utiliser l'opérateur = pour affecter une valeur à une variable.

#### Assigner une variable

// nomVariable = valeur;

prix = 10;

La valeur sur le côté droit de l'expression est affectée à la variable sur le côté gauche de l'expression.

Vous pouvez déclarer une variable et lui affecter une valeur en même temps.

L'exemple de code suivant déclare un int nommé price et attribue la valeur 10.

#### Déclaration et attribution de variables

prix int = 10;

Lorsque vous déclarez une variable, elle contient une valeur aléatoire jusqu'à ce que vous lui affectiez une valeur. Ce comportement était une riche source de bogues dans les programmes C et C ++ qui créaient une variable et l'utilisaient accidentellement comme source d'informations avant de lui donner une valeur. Visual C # ne vous permet pas d'utiliser une variable non attribuée. Vous devez attribuer une valeur à une variable avant de pouvoir l'utiliser; sinon, votre programme risque de ne pas se compiler.

### Variables implicitement typées

Lorsque vous déclarez des variables, vous pouvez également utiliser le mot clé var au lieu de spécifier un type de données explicite tel que int ou string. Lorsque le compilateur voit le mot-clé var, il utilise la valeur affectée à la variable pour déterminer le type.

L'exemple suivant montre comment utiliser le mot clé var pour déclarer une variable.

#### Déclaration d'une variable à l'aide du mot-clé var

Var prix = 20;

Dans cet exemple, la variable de prix est une variable typée implicitement. Cependant, le mot-clé var ne signifie pas que vous pouvez attribuer ultérieurement une valeur d'un type différent à price. Le type de prix est fixe, à peu près de la même manière que si vous l'aviez explicitement déclaré comme une variable entière.

Les variables typées implicitement sont utiles lorsque vous ne connaissez pas, ou qu'il est difficile d'établir explicitement, le type d'une expression que vous souhaitez affecter à une variable.

### Variables d'objet

Lorsque vous déclarez une variable objet, elle n'est initialement pas affectée. Pour utiliser une variable objet, vous devez créer une instance de la classe correspondante, en utilisant l'opérateur new, et l'affecter à la variable objet.

L'opérateur new fait deux choses: il oblige le CLR à allouer de la mémoire pour votre objet, puis il appelle un constructeur pour initialiser les champs de cet objet. La version du constructeur qui s'exécute dépend des paramètres que vous spécifiez pour le nouvel opérateur.

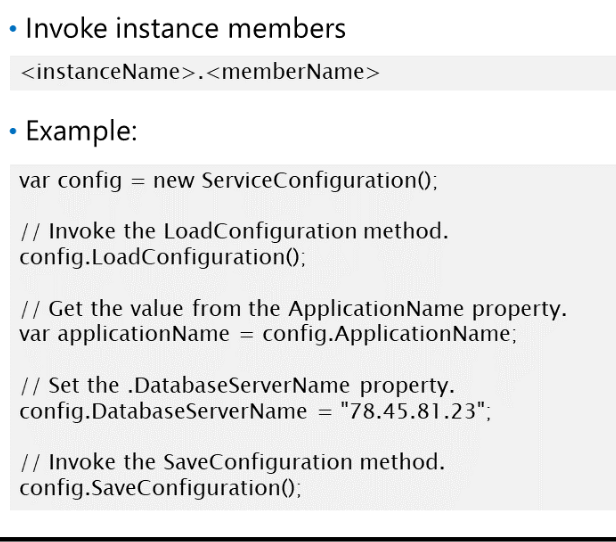
L'exemple de code suivant montre comment créer une instance d'une classe à l'aide de l'opérateur new.

#### Le nouvel opérateur

ServiceConfiguration config = new ServiceConfiguration ();

**Lecture supplémentaire:**Pour plus d'informations sur la déclaration et l'affectation de variables, consultez la page Variables locales implicitement typées (Guide de programmation C #) à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267772>.

## Accès aux membres de type



Pour accéder à un membre d'une instance d'un type, utilisez le nom de l'instance, suivi d'un point, suivi du nom du membre. Ceci est connu sous le nom de notation par points. Tenez compte des règles et instructions suivantes lorsque vous accédez à un membre d'une instance:

|  |  |
| --- | --- |
| • | Pour accéder à une méthode, utilisez des parenthèses après le nom de la méthode. Entre parenthèses, transmettez les valeurs de tous les paramètres requis par la méthode. Si la méthode ne prend aucun paramètre, les parenthèses sont toujours obligatoires. |
| • | Pour accéder à une propriété publique, utilisez le nom de la propriété. Vous pouvez ensuite obtenir la valeur de cette propriété ou définir la valeur de cette propriété. |

L'exemple de code suivant montre comment appeler les membres que la classe ServiceConfiguration expose.

### Invoquer des membres

var config = new ServiceConfiguration ();

// Appel de la méthode LoadConfiguration.

var loadSuccessful = config.LoadConfiguration ();

// Récupère la valeur de la propriété ApplicationName.

var applicationName = config.ApplicationName;

// Définit la propriété .DatabaseServerName.

config.DatabaseServerName = "78.45.81.23";

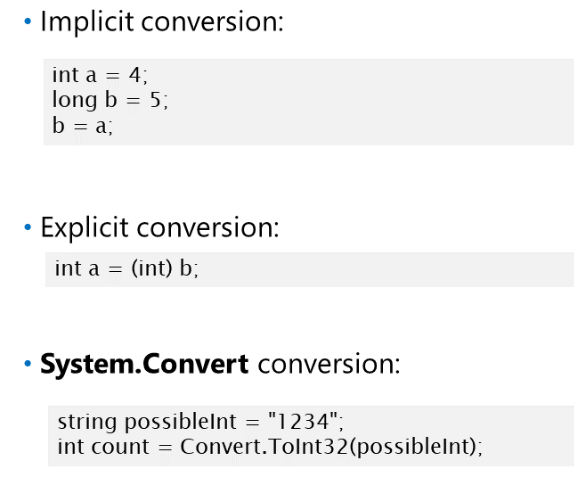
// Appelle la méthode SaveConfiguration.

var saveSuccessful = config.SaveConfiguration ();

**Lecture supplémentaire:**Pour plus d'informations sur l'utilisation des propriétés, consultez la page Propriétés (Guide de programmation C #) à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267773>.

**Lecture supplémentaire:**Pour plus d'informations sur l'utilisation des méthodes, consultez la page Méthodes (Guide de programmation C #) à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267774>.

## Diffusion entre les types de données



Lorsque vous développez une application, vous devrez souvent convertir des données d'un type à un autre, par exemple, lorsqu'une valeur d'un type est affectée à une variable d'un type différent. Considérez le scénario où un utilisateur entre un nombre dans une zone de texte. Pour utiliser ce nombre dans un calcul numérique, vous devrez convertir la valeur de chaîne 99 que vous avez lue dans la zone de texte en valeur entière 99 afin de pouvoir la stocker dans une variable entière. Le processus de conversion d'une valeur d'un type de données en un autre type est appelé conversion de type ou transtypage.

Il existe deux types de conversions dans le .NET Framework:

|  |  |
| --- | --- |
| • | Conversion implicite, qui est automatiquement effectuée par le CLR sur les opérations dont le succès est garanti sans perte d'informations. |
| • | Conversion explicite, qui vous oblige à écrire du code pour effectuer une conversion qui pourrait autrement perdre des informations ou produire une erreur. |

La conversion explicite réduit la possibilité de bogues dans votre code et rend votre code plus efficace. Visual C # interdit les conversions implicites qui perdent de la précision. Cependant, sachez que certaines conversions explicites peuvent donner des résultats inattendus.

### Conversions implicites

Une conversion implicite se produit lorsqu'une valeur est convertie automatiquement d'un type de données à un autre. La conversion ne nécessite aucune syntaxe particulière dans le code source. Visual C # autorise uniquement les conversions implicites sûres, telles que l'élargissement d'un entier.

L'exemple de code suivant montre comment les données sont converties implicitement d'un entier à un long, ce que l'on appelle élargissement.

#### Conversion implicite

int a = 4;

long b;

b = a; // Conversion implicite de int en long.

Cette conversion réussit toujours et n'entraîne jamais de perte d'informations. Cependant, vous ne pouvez pas convertir implicitement une valeur longue en un entier, car cette conversion risque de perdre des informations (la valeur longue peut être en dehors de la plage prise en charge par le type int). Le tableau suivant montre les conversions de type implicites qui sont prises en charge dans Visual C #.

| **From** | **To** |
| --- | --- |
| sbyte | short, int, long, float, double, decimal |
| byte | short, ushort, int, uint, long, ulong, float, double, decimal |
| short | int, long, float, double, decimal |
| ushort | int, uint, long, ulong, float, double, decimal |
| int | long, float, double, decimal |
| uint | long, ulong, float, double, decimal |
| long, ulong | float, double, decimal |
| float | double |
| char | ushort, int, uint, long, ulong, float, double, decimal |

### Conversions explicites

Dans Visual C #, vous pouvez utiliser un opérateur de cast pour effectuer des conversions explicites. Un cast spécifie le type vers lequel effectuer la conversion, entre parenthèses avant le nom de la variable.

L'exemple de code suivant montre comment effectuer une conversion explicite.

#### Conversion explicite

int a;

long b = 5;

a = (int) b; // Conversion explicite de long en int.

Vous ne pouvez effectuer que des conversions significatives de cette manière, telles que la conversion d'un long en un entier. Vous ne pouvez pas utiliser de conversion si le format des données doit changer physiquement, par exemple si vous convertissez une chaîne en entier. Pour effectuer ces types de conversions, vous pouvez utiliser les méthodes de la classe System.Convert.

### Utilisation de la classe System.Convert

La classe System.Convert fournit des méthodes qui peuvent convertir un type de données de base en un autre type de données de base. Ces méthodes portent des noms tels que ToDouble, ToInt32, ToString, etc. Tous les langages qui ciblent le CLR peuvent utiliser cette classe. Vous trouverez peut-être cette classe plus facile à utiliser pour les conversions que les conversions implicites ou explicites, car IntelliSense vous aide à localiser la méthode de conversion dont vous avez besoin.

L'exemple de code suivant convertit une chaîne en un int.

#### Conversions à l'aide de la méthode ToInt32

string possibleInt = "1234";

int count = Convert.ToInt32 (possibleInt);

Certains des types de données intégrés dans Visual C # fournissent également une méthode TryParse, qui vous permet de déterminer si la conversion réussira avant d'effectuer la conversion.

L'exemple de code suivant montre comment convertir une chaîne en int à l'aide de la méthode int.TryParse ().

#### TryParse Conversion

int number = 0;

string numberString = "1234";

if (int.TryParse (numberString, out number))

{

// La conversion a réussi, le nombre est maintenant égal à 1234.

}

autre

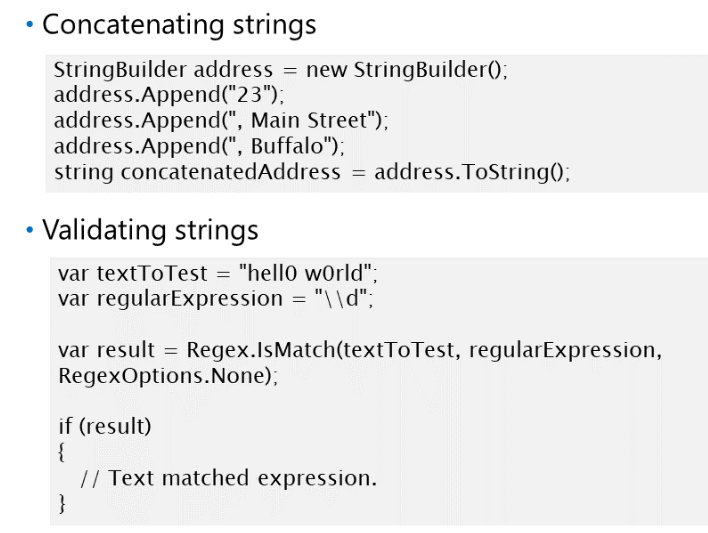
{

// La conversion a échoué, le nombre est désormais égal à 0.

}

**Lecture supplémentaire:**Pour plus d'informations sur la conversion des variables, consultez la page Casting and Type Conversions (C # Programming Guide) à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267775>.

## Manipuler des chaînes



Les chaînes sont un type de données très utile qui vous permet de capturer et de stocker des données alphanumériques.

### Concaténation de chaînes

La concaténation de plusieurs chaînes dans Visual C # est simple à réaliser à l'aide de l'opérateur +. Cependant, cela est considéré comme une mauvaise pratique de codage car les chaînes sont immuables. Cela signifie que chaque fois que vous concaténez une chaîne, vous créez une nouvelle chaîne en mémoire et l'ancienne chaîne est supprimée.

**Noter:**Dans Visual C #, les types de données sont mutables ou immuables. Cela fait référence à leur capacité à changer leurs valeurs intrinsèques. Les types de données immuables ne peuvent pas être modifiés et toute modification qui leur est apportée entraîne la création d'une nouvelle copie de la valeur modifiée à côté de l'ancienne valeur. Les types de données mutables peuvent être modifiés et ne sont pas copiés lorsqu'ils sont modifiés. Par conséquent, lorsque vous concaténez une chaîne, une nouvelle valeur avec la chaîne complète est créée en mémoire, à côté des deux chaînes d'origine.

L'exemple de code suivant crée cinq valeurs de chaîne lors de son exécution.

#### Concaténation à l'aide de l'opérateur +

adresse de chaîne = "23";

adresse = adresse + ", rue principale";

adresse = adresse + ", Buffalo";

Une autre approche consiste à utiliser la classe StringBuilder, qui vous permet de créer une chaîne de manière dynamique et beaucoup plus efficace.

L'exemple de code suivant montre comment utiliser la classe StringBuilder.

#### Concaténation à l'aide de la classe StringBuilder

Adresse StringBuilder = nouveau StringBuilder ();

address.Append ("23");

address.Append (", rue principale");

adresse.Append (", Buffalo");

string concatenatedAddress = adresse.ToString ();

### Validation des chaînes

Lors de l'acquisition d'entrée à partir de l'interface utilisateur d'une application, les données sont souvent fournies sous forme de chaînes que vous devez valider, puis convertir dans un format attendu par votre logique d'application. Par exemple, un contrôle de zone de texte dans une application WPF renverra son contenu sous forme de chaîne, même si un utilisateur a spécifié une valeur entière. Il est important de valider cette entrée afin de minimiser le risque d'erreurs, telles que InvalidCastExceptions.

Les expressions régulières fournissent un mécanisme qui vous permet de valider l'entrée. Le .NET Framework fournit l'espace de noms System.Text.RegularExpressions qui inclut la classe Regex. Vous pouvez utiliser la classe Regex dans vos applications pour tester une chaîne afin de vous assurer qu'elle est conforme aux contraintes d'une expression régulière.

L'exemple de code suivant montre comment utiliser la méthode Regex.IsMatch pour voir si une valeur de chaîne contient des chiffres numériques.

#### Regex.IsMatch, méthode

var textToTest = "hell0 w0rld";

var regularExpression = "\\ d";

l'expression \ d correspondra à tous les caractères numériques.

var result = Regex.IsMatch (textToTest, regularExpression, RegexOptions.None);

if (result)

{

// Expression correspondant au texte.

}

Les expressions régulières fournissent une sélection d'expressions que vous pouvez utiliser pour correspondre à une variété de types de données. Par exemple, l'expression \ d correspondra à tous les caractères numériques.

**Lecture supplémentaire:**Pour plus d'informations sur l'utilisation des expressions régulières, consultez la page Classe Regex à l'adresse<https://aka.ms/moc-20483c-m1-pg3>.

# Leçon 3: Constructions du langage de programmation Visual C #

Lors du développement d'une application, vous devrez souvent exécuter une logique basée sur une condition ou exécuter à plusieurs reprises une section de logique jusqu'à ce qu'une condition soit remplie. Vous souhaiterez peut-être également stocker une collection de données associées dans une seule variable. Visual C # fournit un certain nombre de constructions qui vous permettent de modéliser un comportement complexe dans vos applications.

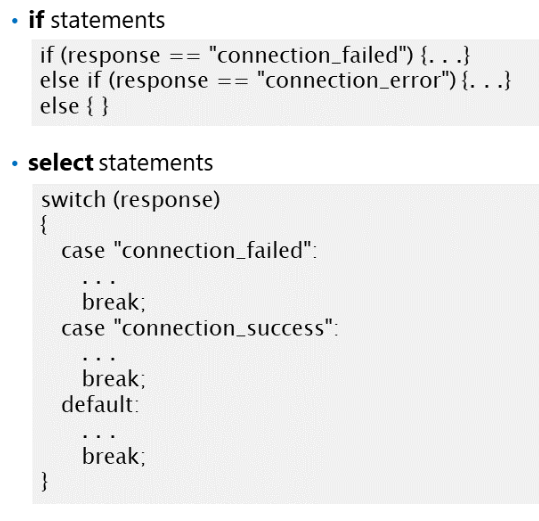
Dans cette leçon, vous apprendrez à implémenter des instructions de décision et d'itération et à stocker des collections de données associées. Vous apprendrez également à structurer l'API de votre application à l'aide d'espaces de noms et à utiliser certaines des fonctionnalités de débogage fournies par Visual Studio.

## Objectifs de la leçon

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Utilisez des instructions conditionnelles. |
| • | Utilisez des instructions d'itération. |
| • | Créez et utilisez des tableaux. |
| • | Décrivez le but des espaces de noms. |
| • | Utilisez des points d'arrêt dans Visual Studio. |

## Implémentation de la logique conditionnelle



La logique d'application a souvent besoin d'exécuter différentes sections de code en fonction de l'état des données dans l'application. Par exemple, si un utilisateur demande à fermer un fichier, il peut lui être demandé s'il souhaite enregistrer des modifications. Si c'est le cas, l'application doit exécuter du code pour enregistrer le fichier. Si ce n'est pas le cas, la logique de l'application peut simplement fermer le fichier. Visual C # utilise des instructions conditionnelles pour déterminer la section de code à exécuter.

L'instruction conditionnelle principale dans Visual C # est l'instruction if. Il existe également une instruction switch que vous pouvez utiliser pour des décisions plus complexes.

### Instructions conditionnelles

Vous utilisez des instructions if pour tester la véracité d'une déclaration. Si l'instruction est vraie, le bloc de code associé à l'instruction if est exécuté, si l'instruction est fausse, le contrôle passe sur le bloc.

Le code suivant montre comment utiliser une instruction if pour déterminer si une chaîne contient la valeur connection\_failed.

#### Instruction if

string response = "….";

if (réponse == "connection\_failed")

{

// Bloc de code à exécuter si la valeur de la variable de réponse est "connection\_failed".

}

**L’instruction if** peuvent avoir des clauses else associées. Le bloc else s'exécute lorsque l'instruction if est fausse.

L'exemple de code suivant montre comment utiliser une instruction if else pour exécuter du code lorsqu'une condition est fausse.

#### Instruction if else

string response = "….";

if (réponse == "connection\_failed")

{

// Le bloc de code s'exécute si la valeur de la variable de réponse est "connection\_failed".

}

else

{

// Le bloc de code s'exécute si la valeur de la variable de réponse n'est pas "connection\_failed".

}

**L’instruction If**  peuvent également avoir des clauses else if associées. Les clauses sont testées dans l'ordre dans lequel elles apparaissent dans le code après l'instruction if. Si l'une des clauses renvoie true, le bloc de code associé à cette instruction est exécuté et le contrôle quitte le bloc de code associé à l'ensemble de la construction if.

L'exemple de code suivant montre comment utiliser une instruction if avec une clause else if.

#### Instruction else if

string response = "….";

if (réponse == "connection\_failed")

{

// Le bloc de code s'exécute si la valeur de la variable de réponse est "connection\_failed".

}

else if (réponse == "connection\_error")

{

// Le bloc de code s'exécute si la valeur de la variable de réponse est "connection\_error".

}

else

{

// Le bloc de code s'exécute si la valeur de la variable de réponse n'est pas "connection\_failed" ou "connection\_error".

}

#### 

### Instructions de sélection

S'il y a trop d'instructions if / else, le code peut devenir compliqué et difficile à suivre. Dans ce scénario, une meilleure solution consiste à utiliser une instruction switch. L'instruction switch remplace simplement plusieurs instructions if / else.

L'exemple suivant montre comment utiliser une instruction switch pour remplacer une collection de clauses elseif.

#### Instruction switch

string response = "….";

switch (response)

{

case "connection\_failed":

// Block of code executes if the value of response is "connection\_failed".

break;

case "connection\_success":

// Block of code executes if the value of response is "connection\_success".

break;

case "connection\_error":

// Block of code executes if the value of response is "connection\_error".

break;

default:

// Block executes if none of the above conditions are met.

break;

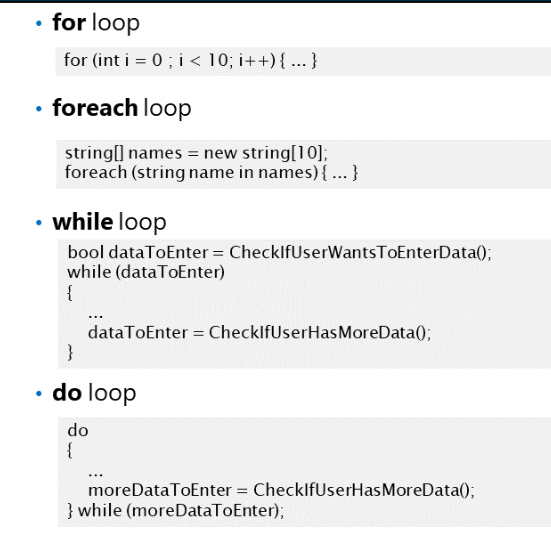
}

Dans chaque instruction case, notez le mot-clé break. Cela amène le contrôle à sauter à la fin du commutateur après le traitement du bloc de code. Un cas de commutateur doit se terminer par une instruction jump, telle que break, return ou goto. Si vous omettez l'instruction jump, votre code ne sera pas compilé.

Notez qu'il existe un bloc étiqueté par défaut:. Ce bloc de code s'exécutera lorsqu'aucun des autres blocs ne correspond.

**Lecture supplémentaire:**Pour plus d'informations sur les instructions de sélection, consultez la page Instructions de sélection (référence C #) à l'adresse<https://aka.ms/moc-20483c-m1-pg4>.

## Implémentation de la logique d'itération



L'itération offre un moyen pratique d'exécuter plusieurs fois un bloc de code. Par exemple, itérer sur une collection d'éléments dans un tableau ou simplement exécuter une fonction plusieurs fois. Visual C # fournit un certain nombre de constructions standard appelées boucles que vous pouvez utiliser pour implémenter la logique d'itération.

**Les boucles for**

La boucle for exécute un bloc de code à plusieurs reprises jusqu'à ce que l'expression spécifiée soit évaluée à false. Vous pouvez définir une boucle for comme suit.

for ([initialiseurs]; [expression]; [itérateurs])

{

[corps]

}

Lorsque vous utilisez une boucle for, vous initialisez d'abord une valeur en tant que compteur. Sur chaque boucle, vous vérifiez que la valeur du compteur est dans la plage pour exécuter la boucle for, et si c'est le cas, exécutez le corps de la boucle.

L'exemple de code suivant montre comment utiliser une boucle for pour exécuter un bloc de code 10 fois.

#### Boucle For

for (int i = 0 ; i < 10; i++)

{

// Code to execute.

}

Dans cet exemple, i = 0; est l'initialiseur, i <10; est l'expression, et i ++; est l'itérateur.

**Boucle For Each**

Bien qu'une boucle for soit facile à utiliser, elle peut être difficile à faire. Par exemple, lors d'une itération sur une collection ou un tableau, vous devez savoir combien d'éléments la collection ou le tableau contient. Dans de nombreux cas, c'est simple, mais parfois il peut être facile de se tromper. Par conséquent, il est parfois préférable d'utiliser une boucle foreach.

L'exemple de code suivant montre comment utiliser une boucle foreach pour itérer un tableau de chaînes.

#### boucle foreach

**string[] names = new string[10];**

**// Process each name in the array.**

**foreach (string name in names)**

**{**

**// Code to execute.**

**}**

**Les boucles While**

Une boucle while vous permet d'exécuter un bloc de code alors qu'une condition donnée est vraie. Par exemple, vous pouvez utiliser une boucle while pour traiter l'entrée utilisateur jusqu'à ce que l'utilisateur indique qu'il n'a plus de données à saisir.

L'exemple de code suivant montre comment utiliser une boucle while.

#### Boucle while

bool dataToEnter = CheckIfUserWantsToEnterData();

while (dataToEnter)

{

// Process the data.

dataToEnter = CheckIfUserHasMoreData();

}

**Boucle do**

Une boucle do est très similaire à une boucle while, à l'exception qu'une boucle do s'exécutera toujours au moins une fois. Alors que si la condition n'est pas initialement remplie, une boucle while ne s'exécutera jamais. Par exemple, vous pouvez utiliser une boucle do si vous savez que ce code ne s'exécutera qu'en réponse à une demande utilisateur de saisir des données. Dans ce scénario, vous savez que l'application devra traiter au moins une donnée et peut donc utiliser une boucle do.

L'exemple de code suivant montre comment utiliser une boucle do.

#### Boucle do

**do**

**{**

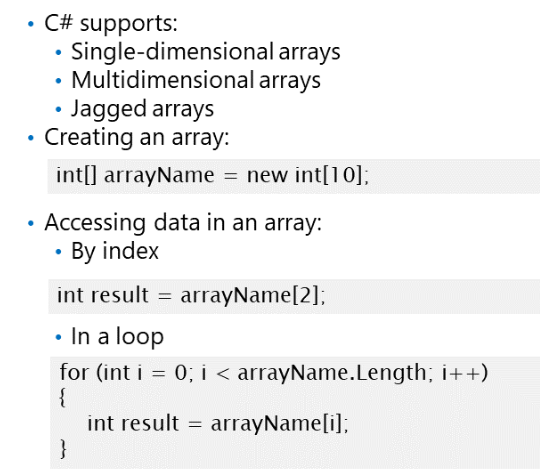
**// Process the data.**

**moreDataToEnter = CheckIfUserHasMoreData();**

**} while (moreDataToEnter);**

**Lecture supplémentaire:**Pour plus d'informations sur les boucles, consultez la page Instructions d'itération (référence C #) à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267778>.

## Création et utilisation de tableaux



Un tableau est un ensemble d'objets regroupés et gérés comme une unité. Vous pouvez considérer un tableau comme une séquence d'éléments, qui sont tous du même type. Vous pouvez créer des tableaux simples qui ont une dimension (une liste), deux dimensions (un tableau), trois dimensions (un cube), etc. Les tableaux dans Visual C # présentent les fonctionnalités suivantes:

|  |  |
| --- | --- |
| • | Chaque élément du tableau contient une valeur. |
| • | Les tableaux sont indexés à zéro, c'est-à-dire que le premier élément du tableau est l'élément 0. |
| • | La taille d'un tableau est le nombre total d'éléments qu'il peut contenir. |
| • | Les tableaux peuvent être unidimensionnels, multidimensionnels ou irréguliers. |
| • | Le rang d'un tableau est le nombre de dimensions du tableau. |

Les tableaux d'un type particulier ne peuvent contenir que des éléments de ce type. Si vous devez manipuler un ensemble d'objets ou de types valeur différents, envisagez d'utiliser l'un des types de collection définis dans l'espace de noms System.Collections.

### Création de tableaux

Lorsque vous déclarez un tableau, vous spécifiez le type de données qu'il contient et un nom pour le tableau. Déclarer un tableau amène le tableau dans la portée, mais n'alloue pas réellement de mémoire pour lui. Le CLR crée physiquement le tableau lorsque vous utilisez le nouveau mot-clé. À ce stade, vous devez spécifier la taille du tableau.

La liste suivante décrit comment créer des tableaux unidimensionnels, multidimensionnels et irréguliers:

|  |  |
| --- | --- |
| • | Tableaux unidimensionnels.  Pour déclarer un tableau unidimensionnel, vous spécifiez le type d'éléments dans le tableau et utilisez des crochets, [] pour indiquer qu'une variable est un tableau. Plus tard, vous spécifiez la taille du tableau lorsque vous allouez de la mémoire pour le tableau à l'aide du mot clé new. La taille d'un tableau peut être n'importe quelle expression entière. L'exemple de code suivant montre comment créer un tableau unidimensionnel d'entiers avec des éléments de zéro à neuf.  int [] arrayName = new int [10]; |
| • | Tableaux multidimensionnels.  Un tableau peut avoir plusieurs dimensions. Le nombre de dimensions correspond au nombre d'index utilisés pour identifier un élément individuel dans le tableau. Vous pouvez spécifier jusqu'à 32 dimensions, mais vous en aurez rarement besoin de plus de trois. Vous déclarez une variable de tableau multidimensionnel tout comme vous déclarez un tableau unidimensionnel, mais vous séparez les dimensions à l'aide de virgules. L'exemple de code suivant montre comment créer un tableau d'entiers avec trois dimensions.  int [,,] arrayName = new int [10,10,10]; |
| • | Tableaux dentelés.  Un tableau irrégulier est simplement un tableau de tableaux, et la taille de chaque tableau peut varier. Les tableaux en dents de scie sont utiles pour modéliser des structures de données clairsemées où vous ne voudrez peut-être pas toujours allouer de la mémoire pour chaque élément s'il n'est pas destiné à être utilisé. L'exemple de code suivant montre comment déclarer et initialiser un tableau dentelé. Notez que vous devez spécifier la taille du premier tableau, mais vous ne devez pas spécifier la taille des tableaux contenus dans ce tableau. Vous allouez de la mémoire à chaque tableau dans un tableau en dents de scie séparément, en utilisant le mot-clé new. |

**Accès aux données dans une baie**

int[][] jaggedArray = new int[10][];

jaggedArray[0] = new Type[5]; // Can specify different sizes.

jaggedArray[1] = new Type[7];

...

jaggedArray[9] = new Type[21];

Vous pouvez accéder aux données d'un tableau de plusieurs manières, par exemple en spécifiant l'index d'un élément spécifique dont vous avez besoin ou en parcourant toute la collection et en renvoyant chaque élément dans l'ordre.

L'exemple de code suivant utilise un index pour accéder à l'élément à l'index deux.

#### Accès aux données par index

int [] oldNumbers = {1, 2, 3, 4, 5};

nombre entier = oldNumbers [2];

**Noter:**

Les tableaux sont indexés à zéro, de sorte que le premier élément de toute dimension d'un tableau est à l'index zéro. Le dernier élément d'une dimension est à l'index N-1, où N est la taille de la dimension. Si vous essayez d'accéder à un élément en dehors de cette plage, le CLR lève une exception IndexOutOfRangeException.

Vous pouvez parcourir un tableau en utilisant une boucle for. Vous pouvez utiliser la propriété Length du tableau pour déterminer quand arrêter la boucle.

L'exemple de code suivant montre comment utiliser une boucle for pour parcourir un tableau.

#### Itération sur un tableau

**int[] oldNumbers = { 1, 2, 3, 4, 5 };**

**for (int i = 0; i < oldNumbers.Length; i++)**

**{**

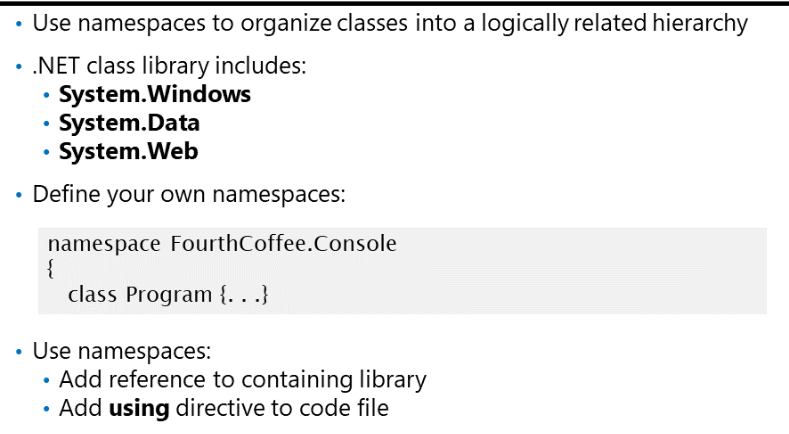
**int number = oldNumbers[i];**

**...**

**}**

**Lecture supplémentaire:**Pour plus d'informations sur les tableaux, consultez la page Tableaux (Guide de programmation C #) à l'adresse<http://go.microsoft.com/fwlink/?LinkID=267779>.

## Référencer les espaces de noms



Le Microsoft .NET Framework se compose de nombreux espaces de noms qui organisent ses classes dans des hiérarchies liées logiquement. Vous pouvez utiliser des espaces de noms dans vos propres applications pour organiser de la même manière vos classes en hiérarchies.

Les espaces de noms fonctionnent à la fois comme un système interne pour organiser votre application et comme un moyen externe d'éviter les conflits de noms entre votre code et d'autres applications. Chaque espace de noms contient des types que vous pouvez utiliser dans votre programme, tels que des classes, des structures, des énumérations, des délégués et des interfaces. Étant donné que différentes classes peuvent avoir le même nom, vous utilisez des espaces de noms pour différencier la même classe nommée en deux hiérarchies différentes afin d'éviter les problèmes d'interopérabilité.

### Espaces de noms de la bibliothèque de classes .NET Framework

L'espace de noms le plus important du .NET Framework est l'espace de noms System, qui contient les classes que la plupart des applications utilisent pour interagir avec le système d'exploitation. Quelques-uns des espaces de noms fournis par .NET Framework via l'espace de noms System sont répertoriés dans le tableau suivant:

| **Espace de noms** | **Définition** |
| --- | --- |
| Système.Windows | Fournit les classes utiles pour créer des applications WPF. |
| System.IO | Fournit des classes pour lire et écrire des données dans des fichiers. |
| System.Data | Fournit des classes pour l'accès aux données. |
| System.Web | Fournit des classes utiles pour créer des applications Web. |

### Espaces de noms définis par l'utilisateur

Les espaces de noms définis par l'utilisateur sont des espaces de noms définis dans votre code. Il est recommandé de définir toutes vos classes dans des espaces de noms. L'environnement Visual Studio suit cette recommandation en utilisant le nom de votre projet comme espace de noms de niveau supérieur dans un projet.

L'exemple de code suivant montre comment définir un espace de noms avec le nom FourthCoffee.Console, qui contient la classe Program.

#### Définition d'un espace de noms

namespace FourthCoffee.Console

{

class Program

{

static void Main(string[] args)

{

}

}

}

### Utilisation des espaces de noms

Lorsque vous créez un projet Visual C # dans Visual Studio, les assemblys de classe de base les plus courants sont déjà référencés. Toutefois, si vous devez utiliser un type qui se trouve dans un assemblage qui n'est pas déjà référencé dans votre projet, vous devrez ajouter une référence à l'assemblage à l'aide de la boîte de dialogue Ajouter une référence. Ensuite, en haut de votre fichier de code, vous répertoriez les espaces de noms que vous utilisez dans ce fichier, précédés de la directive using. La directive using indique à votre application d'importer les types dans l'espace de noms pour leur permettre d'être appelés directement, comme s'ils existaient dans l'espace de noms actuel.

L'exemple de code suivant montre comment importer l'espace de noms System et utiliser la classe Console.

#### Importer un espace de noms

using System;

…

Console.WriteLine("Hello, World");

Vous pouvez appeler n'importe quel type dans un assembly référencé à condition de spécifier le nom complet de ce type, y compris son espace de noms. C'est ce que l'on appelle le nom qualifié complet du type.

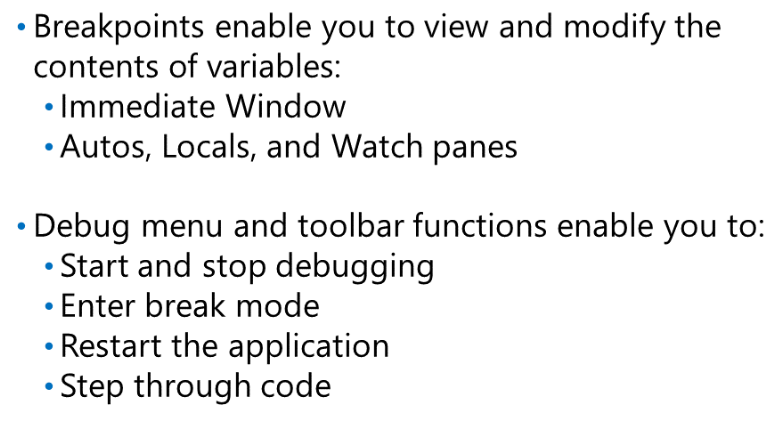
L'exemple de code suivant montre comment utiliser la classe Console sans importer l'espace de noms système.

#### Appeler la console par son nom complet

System.Console.WriteLine ("Bonjour, Monde");

**Lecture supplémentaire:**Pour plus d'informations sur les espaces de noms, consultez la page d'espace de noms (référence C #) à l'adresse<https://aka.ms/moc-20483c-m1-pg2>.

## Utilisation de points d'arrêt dans Visual Studio 2017



Le débogage est une partie essentielle du développement d'applications. Vous remarquerez peut-être des erreurs lorsque vous écrivez du code, mais certaines erreurs, en particulier les erreurs de logique, ne peuvent se produire que dans des circonstances que vous ne prévoyez pas. Les utilisateurs peuvent vous signaler ces erreurs et vous devrez les corriger.

Visual Studio 2017 fournit plusieurs outils pour vous aider à déboguer le code. Vous pouvez les utiliser pendant que vous développez du code, pendant une phase de test ou après la publication de l'application. Vous utiliserez les outils de la même manière quelles que soient les circonstances. Vous pouvez exécuter une application avec ou sans le débogage activé. Lorsque le débogage est activé, votre application est dite en mode débogage.

### Utilisation des points d'arrêt

Si vous connaissez l'emplacement approximatif du problème dans votre code, vous pouvez utiliser un point d'arrêt pour que le débogueur Visual Studio entre en mode arrêt avant d'exécuter une ligne de code spécifique. Cela vous permet d'utiliser les outils de débogage pour revoir ou modifier l'état de votre application pour vous aider à corriger le bogue. Pour ajouter un point d'arrêt à une ligne de code, dans le menu Déboguer, cliquez sur Basculer le point d'arrêt.

Lorsque vous êtes en mode pause, vous pouvez survoler les noms de variables pour afficher leur valeur actuelle. Vous pouvez également utiliser la fenêtre Exécution et les volets Autos, Locals et Watch pour afficher et modifier le contenu des variables.

### Utilisation des contrôles de débogage

Après avoir visualisé ou modifié des variables en mode pause, vous souhaiterez probablement parcourir les lignes de code suivantes dans votre application. Vous souhaiterez peut-être simplement exécuter le reste de l'application ou vous souhaiterez peut-être exécuter une ligne de code à la fois. Visual Studio fournit une variété de commandes dans le menu Déboguer qui vous permettent de faire cela et plus encore. Le tableau suivant répertorie les éléments clés du menu Déboguer et de la barre d'outils Débogage, ainsi que les raccourcis clavier correspondants pour naviguer dans votre code.

| **Élément du menu** | **Bouton de la barre d'outils** | **Raccourci clavier** | **Description** |
| --- | --- | --- | --- |
| Démarrer le débogage | Démarrer / continuer | F5 | Ce bouton est disponible lorsque votre application n'est pas en cours d'exécution et lorsque vous êtes en mode pause. Il démarrera votre application en mode Débogage ou reprendra l'application si vous êtes en mode pause. |
| Tout casser | Tout casser | Ctrl + Alt + Pause | Ce bouton met le traitement de l'application en pause et passe en mode pause. Le bouton est disponible lorsqu'une application est en cours d'exécution. |
| Arrêter le débogage | Arrêter | Maj + F5 | Ce bouton arrête le débogage. Il est disponible lorsqu'une application est en cours d'exécution ou en mode pause. |
| Redémarrage | Redémarrage | Ctrl + Maj + F5 | Ce bouton équivaut à arrêter suivi de démarrage. Cela entraînera le redémarrage de votre application depuis le début. Il est disponible lorsqu'une application est en cours d'exécution ou en mode pause. |
| Entrer dans | Entrer dans | F11 | Ce bouton est utilisé pour accéder aux appels de méthode. |
| Enjamber | Enjamber | F10 | Ce bouton est utilisé pour parcourir les appels de méthode. |
| Sors | Sors | Maj + F11 | Ce bouton est utilisé pour exécuter le code restant dans la méthode et revenir à l'instruction suivante dans la méthode appelante. |

**Lecture supplémentaire:**Pour plus d'informations sur le débogage, consultez la page Débogage dans Visual Studio à l'adresse<https://aka.ms/moc-20483c-m1-pg6>.

# Démonstration: développement du laboratoire d'application d'inscription aux classes

Dans cette démonstration, vous découvrirez les tâches que vous effectuerez dans le laboratoire pour ce module.

**Démonstration étapes**

Vous trouverez les étapes dans la section Démonstration: Développement du laboratoire d'application d'inscription aux cours sur la page suivante:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD01_DEMO.md>.

## Laboratoire: Développer l'application d'inscription aux classes

**Scénario**

Vous êtes un développeur Visual C # travaillant pour une société de développement de logiciels qui écrit des applications pour The School of Fine Arts, une école primaire pour enfants surdoués.

Les administrateurs de l'école ont besoin d'une application qu'ils peuvent utiliser pour inscrire des élèves dans une classe. L'application doit permettre à un administrateur d'ajouter et de supprimer des étudiants des classes, ainsi que de mettre à jour les détails des étudiants.

Il vous a été demandé d'écrire le code qui implémente la logique métier de l'application.

**Noter:**Pendant les travaux pratiques des deux premiers modules de ce cours, vous écrirez le code de cette demande d'inscription au cours.

Lorsque l'École des beaux-arts vous demande d'étendre les fonctionnalités de l'application, vous réalisez que vous devrez tester une preuve de concept et obtenir les commentaires des clients avant d'écrire l'application finale, donc dans le laboratoire du module 3, vous commencerez à développer une application prototype. et continuez avec ceci jusqu'à la fin du module 8.

Dans le laboratoire du module 9, après avoir obtenu l'approbation de l'application finale, vous développerez l'interface utilisateur pour la version de production de l'application, sur laquelle vous travaillerez pendant le reste du cours.

## Objectifs

Après avoir terminé cet atelier, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Écrivez du code Visual C # qui implémente la logique nécessaire pour modifier les détails d'un étudiant. |
| • | Écrivez du code Visual C # qui implémente la logique nécessaire pour ajouter de nouveaux étudiants. |
| • | Écrivez du code Visual C # qui implémente la logique nécessaire pour supprimer des étudiants d'une classe. |
| • | Effectuez des transformations de données simples pour afficher des informations. |

### Configuration du laboratoire

Durée estimée: 105 minutes

Vous trouverez les étapes de haut niveau sur la page suivante:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD01_LAB_MANUAL.md>.

Vous trouverez les étapes détaillées sur la page suivante:<https://github.com/MicrosoftLearning/20483-Programming-in-C-Sharp/blob/master/Instructions/20483C_MOD01_LAK.md>.

### Exercice 1: Implémentation de la fonctionnalité d'édition pour la liste des étudiants

#### Scénario

Dans cet exercice, vous allez écrire le code qui permet à un administrateur utilisant l'application de modifier les détails d'un étudiant.

Une liste d'étudiants s'affiche dans l'interface utilisateur de l'application. Lorsque l'utilisateur sélectionne un élève puis appuie sur une touche du clavier, vous vérifiez si la touche sur laquelle il a appuyé était Entrée. S'ils ont appuyé sur Entrée, vous écrirez le code pour afficher les détails de l'étudiant dans un formulaire séparé, que l'utilisateur peut utiliser pour modifier les détails. Lorsque l'utilisateur ferme le formulaire, vous recopiez les détails mis à jour dans la zone de liste affichant la liste des étudiants. Enfin, vous exécuterez l'application pour vérifier que votre code fonctionne comme prévu, puis vous utiliserez les outils de débogage pour examiner le code pendant son exécution.

**Résultat**: Une fois cet exercice terminé, les utilisateurs pourront modifier les détails d'un élève.

### Exercice 2: Implémentation de la fonctionnalité d'insertion pour la liste des étudiants

## Scénario

Dans cet exercice, vous allez écrire un code qui permet à un administrateur utilisant l'application d'ajouter un nouvel étudiant à la liste des étudiants.

Une liste d'étudiants s'affiche dans l'interface utilisateur de l'application. Lorsque l'utilisateur appuie sur une touche du clavier, vous vérifiez si la touche sur laquelle il a appuyé était Insérer. S'ils ont appuyé sur Insérer, vous écrirez un code pour afficher un formulaire dans lequel l'utilisateur peut entrer les détails d'un nouvel élève, y compris son prénom, son nom et sa date de naissance. Lorsque l'utilisateur ferme le formulaire, vous ajoutez le nouvel étudiant à la liste des étudiants et affichez les détails dans la zone de liste. Enfin, vous exécuterez l'application pour vérifier que votre code fonctionne comme prévu.

**Résultat**: Une fois cet exercice terminé, les utilisateurs pourront ajouter de nouveaux étudiants à une classe.

### Exercice 3: Implémentation de la fonctionnalité de suppression pour la liste des étudiants

# Scénario

Dans cet exercice, vous allez écrire un code permettant à un administrateur de supprimer un étudiant de la liste des étudiants.

Une liste d'étudiants s'affiche dans l'interface utilisateur de l'application. Si l'utilisateur sélectionne un élève et appuie ensuite sur une touche du clavier, vous vérifierez si la touche sur laquelle il a appuyé était Supprimer. S'ils ont appuyé sur Supprimer, vous écrirez un code pour inviter l'utilisateur à confirmer qu'il souhaite supprimer l'élève sélectionné de la classe. Si tel est le cas, l'élève sera supprimé de la liste des élèves de la classe appropriée, sinon rien ne change. Enfin, vous exécuterez l'application pour vérifier que votre code fonctionne comme prévu.

**Résultat**: Une fois cet exercice terminé, les utilisateurs pourront supprimer des étudiants des cours.

### Exercice 4: Affichage de l'âge d'un élève

#### Scénario

Dans cet exercice, vous allez mettre à jour l'application pour afficher l'âge d'un élève au lieu de sa date de naissance.

Vous écrirez du code dans la classe AgeConverter qui est liée à la colonne de la grille affichant l'âge des élèves. Dans ce cours, vous allez écrire du code pour calculer la différence entre la date actuelle et la date de naissance de l'élève, puis convertir cette valeur en années. Ensuite, vous exécuterez l'application pour vérifier que la colonne Âge affiche désormais l'âge en années au lieu de la date de naissance.

**Résultat**: Une fois cet exercice terminé, l'application affichera l'âge de l'élève en années.

**Revue du module et points à retenir**

Dans ce module, vous avez découvert certaines des fonctionnalités de base fournies par .NET Framework et Microsoft Visual Studio®. Vous avez également découvert certaines des constructions de base Visual C # ® qui vous permettent de commencer à développer des applications .NET Framework.

**Question (s) de révision**