

Introduction à C# et Visual Studio

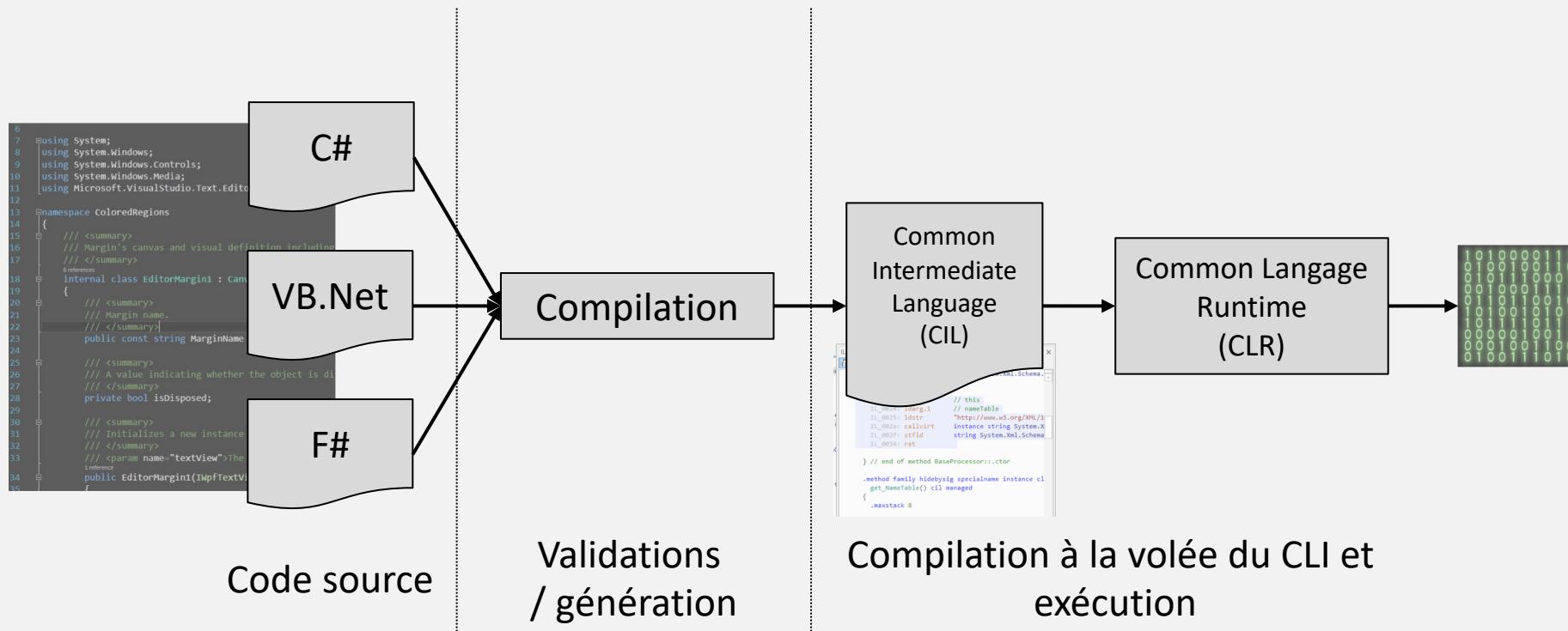
Objectifs

- Introduire le choix du langage
- Accéder à votre environnement de travail
- Créer un premier programme
- Premiers éléments C# et correspondances avec le pseudo-code

C# - Description

- C# est le langage que nous allons utiliser dans le cours
- C# est un des langages offerts dans la plateforme .Net de Microsoft
- Pourquoi C# :
 - Fait partie des langages de programmation structuré fortement typé
 - Facile et plus rapide à rédiger que d'autres langages plus bas niveau
 - Mémoire gérée (généralement) automatiquement
 - Utilisé à Québec
- Multiplateformes : Windows, Unix, Linux
- Microsoft mais Open Source !

C# - Flux : du code à l'application

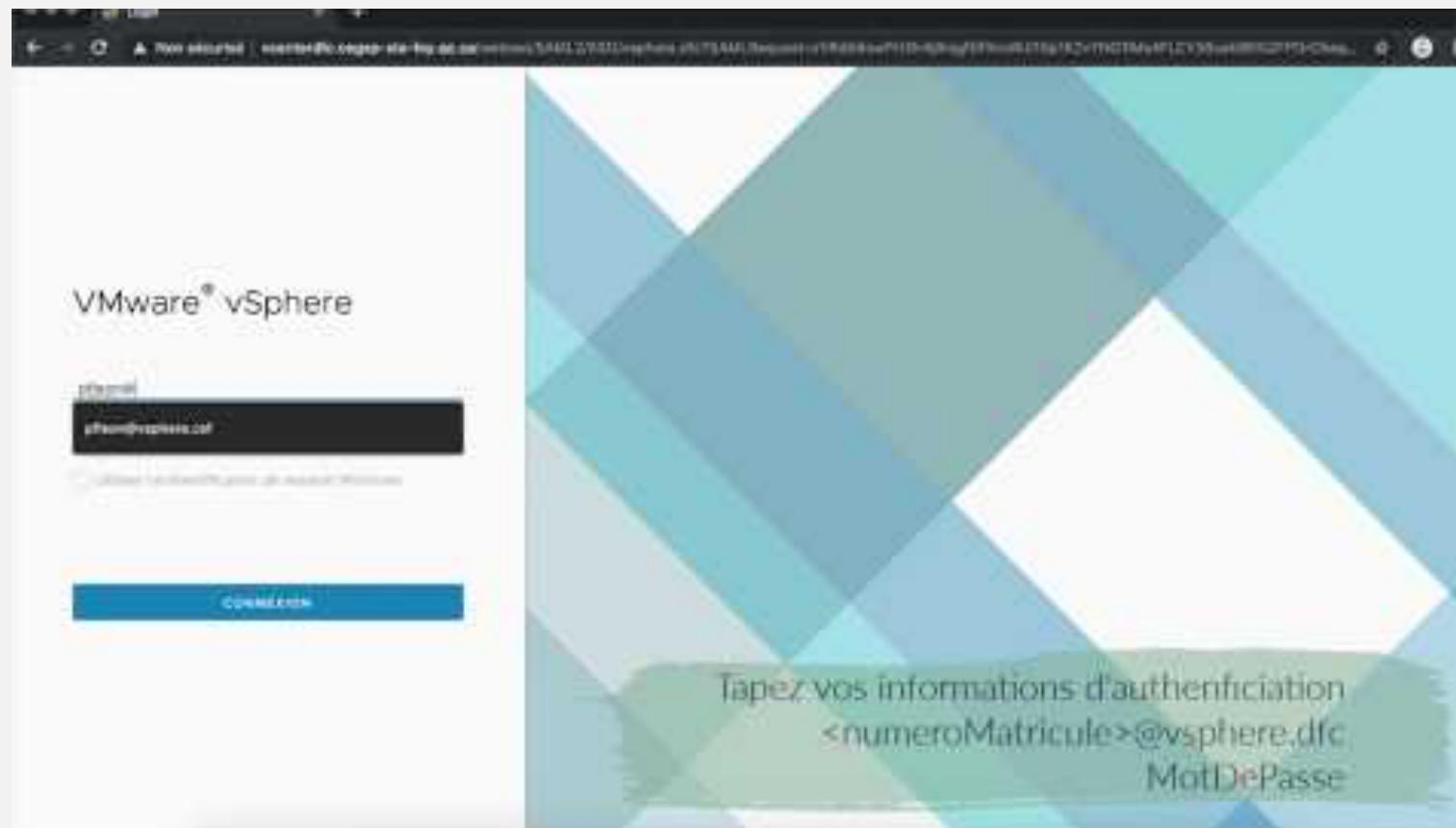


Connexion sur vos machines virtuelles

Accès : <https://vcenterdfc.cegep-ste-foy.qc.ca/>

Compte : <numéroMatricule>@vsphere.dfc

Mot de passe : Sol&i01



<https://youtu.be/7SrpLeLba-o>

Un premier projet C#

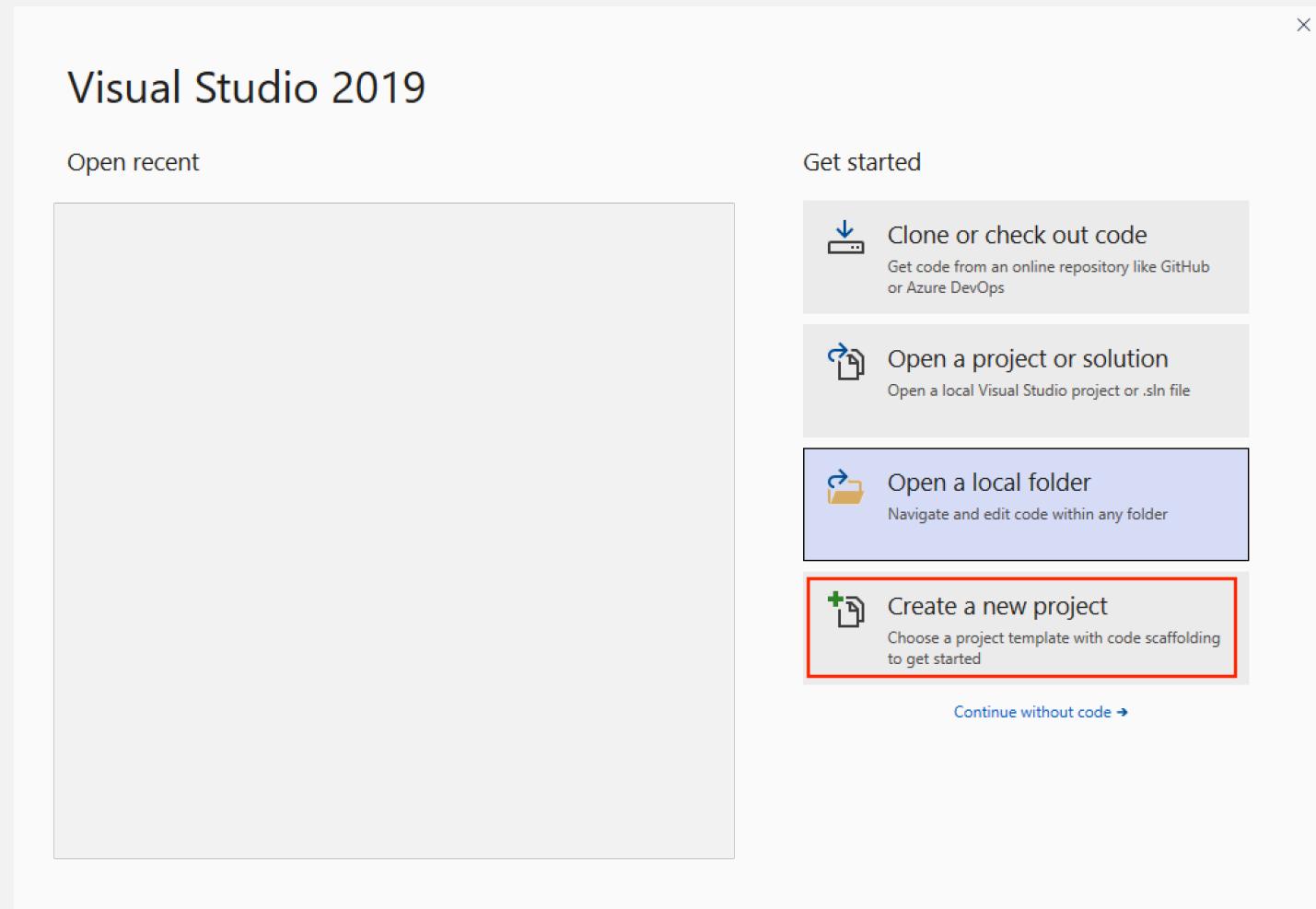
Compte : <numéroMatricule>@cegep-ste-foy.qc.ca
Mot de passe : <votreMotDePasseOmnivox>

UN PREMIER PROJET C# AVEC
VISUAL STUDIO 2019

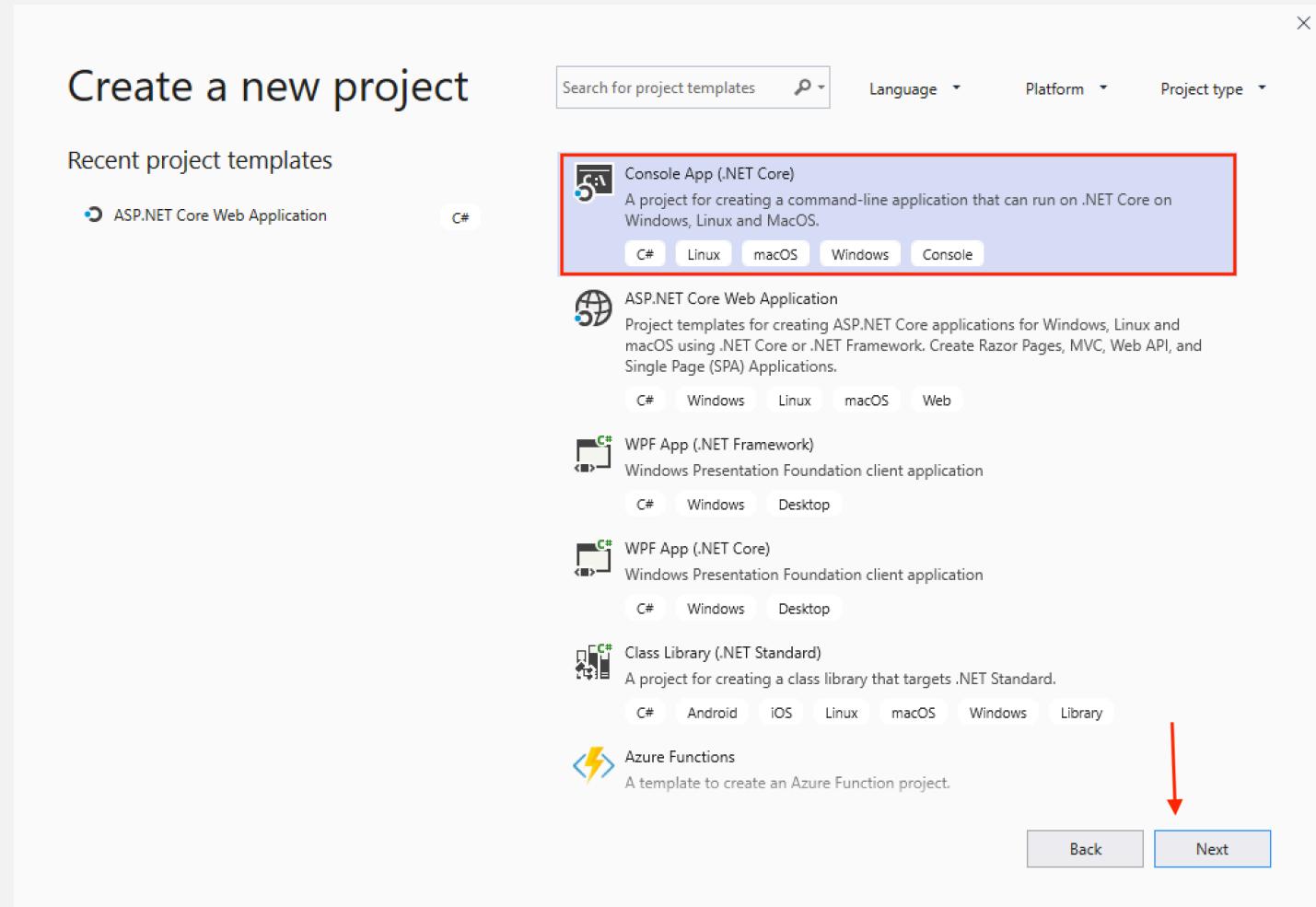
CÉGEP DE SAINTE FOY - PFL

<https://youtu.be/GFc9zHkl3HU>

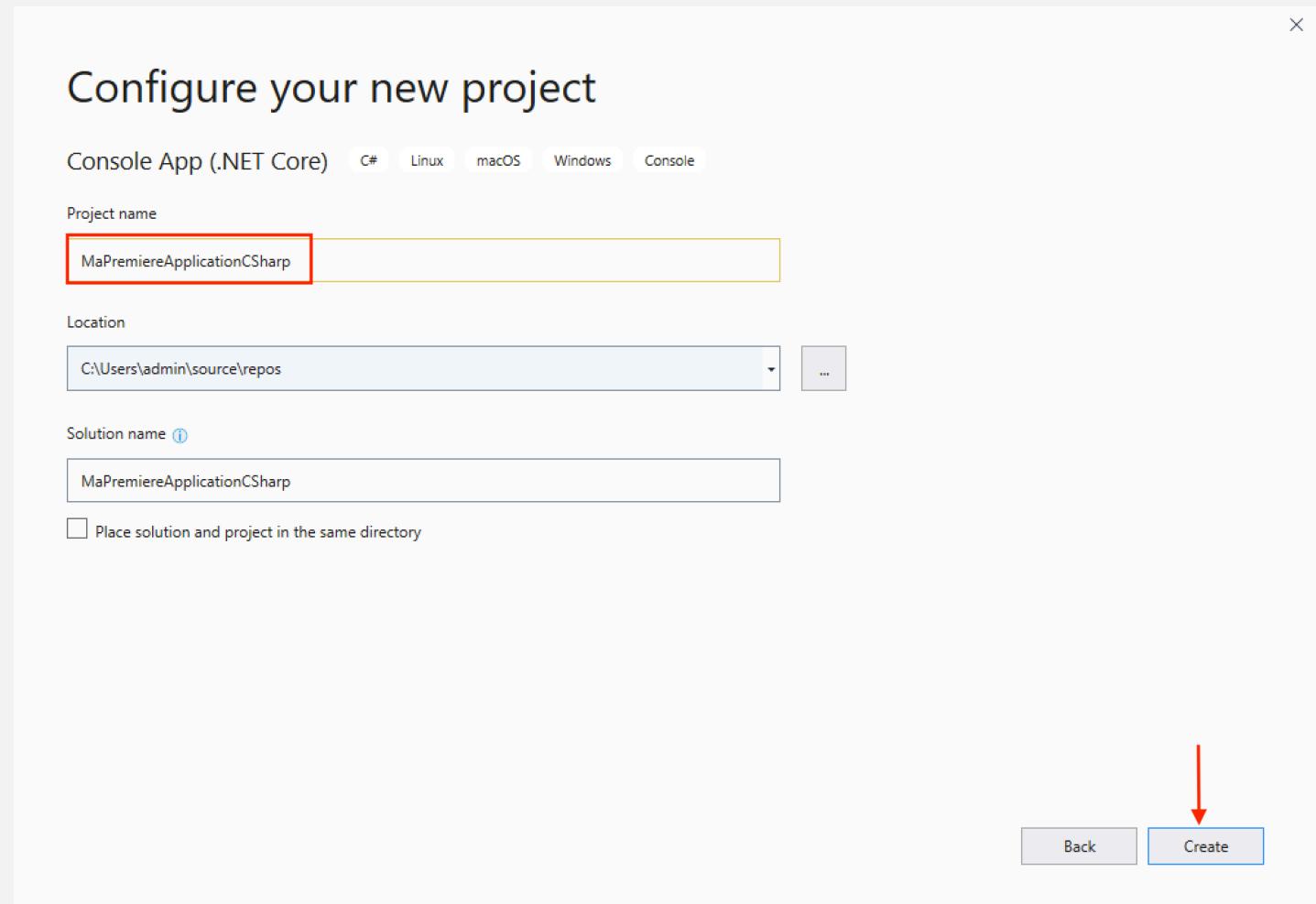
Création d'une première application



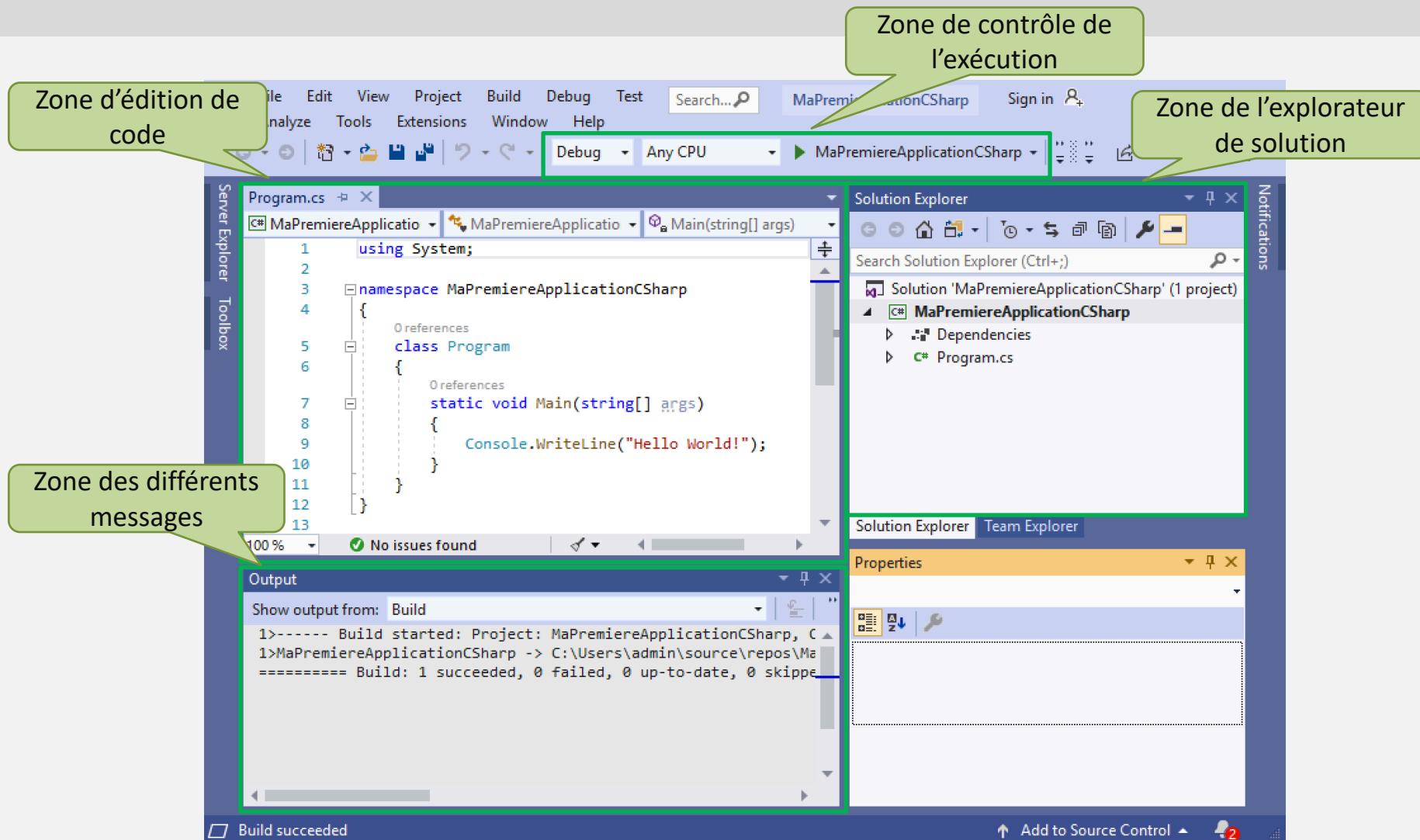
Création d'une première application



Création d'une première application



Création d'une première application



Code du gabarit – Hello world

```
using System;  
namespace MaPremiereApplicationCSharp  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

On spécifie qu'on veut utiliser tout ce qui appartient à l'espace de nom* « System », ici pour pouvoir accéder à « *Console* »

Tout ce qui suit l'accolade « { » jusqu'à l'accolade fermante correspondante sera « rangé » dans l'espace de nom* « *MaPremiereApplicationCSharp* »

La classe** « *Program* » est déclarée dans la plupart des gabarits de projets.

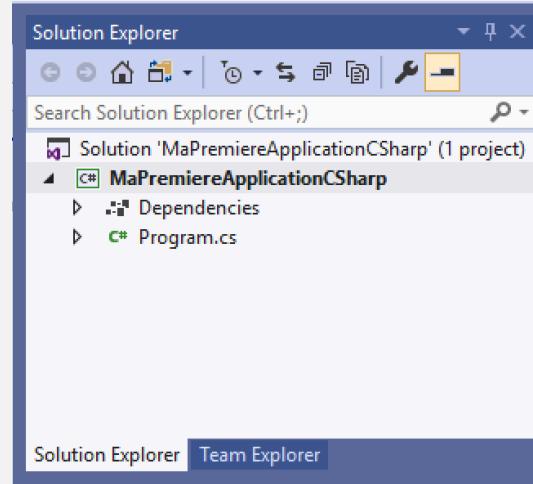
La méthode « *Main* » correspond à la fonction « *Principale* » vue en algo : c'est ce qui est exécuté en premier

Utilisation de la méthode « *WriteLine* » de la classe « *Console* » : Écriture du texte « Hello World! » sur la console texte et ajout d'un retour chariot. La méthode « *Write* » fonctionne de la même façon sans le retour chariot

* Espace de nom : genre de dossier / répertoire où l'on range du code

** Classe : nous verrons cette notion plus tard dans le cours, pour le moment, nous allons nous contenter d'utiliser la classe « *Program* » pour y mettre nos fonctions qui sont en réalité ici des « méthodes statiques » (ici nous admettons la syntaxe, vous y reviendrez plus tard)

Visual Studio – Explorateur de solution



- Une solution peut contenir plusieurs projets
- Chaque projet contient plusieurs fichiers source



- « Debug/Release » correspond au mode de compilation/exécution choisi
- Le bouton lecture permet d'exécuter le programme

Quelques raccourcis pour Visual Studio

Édition

Raccourci	Effet
[Ctrl] + s	Enregistrer le fichier courant
[Ctrl] + c	Copier
[Ctrl] + x	Couper
[Ctrl] + v	Coller
[Ctrl] + .	Accéder aux actions rapides et corrections
[Ctrl] + [espace]	Accéder à l'auto-complétion : choix avec haut / bas et [tab] / [enter] / [.] pour valider
[Ctrl] + k, [Ctrl] + d	Indenter le fichier courant
[Ctrl] + k, [Ctrl] + c	Commenter le texte sélectionné
[Ctrl] + k, [Ctrl] + u	Décommenter le texte sélectionné
[Ctrl] + k, [Ctrl] + s	Entourer la sélection de ...

Compilation

Raccourci	Effet
[Ctrl] + b	Construire la solution

Exécution / débug

Raccourci	Effet
[F5]	Exécuter le code / reprise d'exécution
[F9]	Ajouter/Supprimer point d'arrêt
[F10]	Passer à l'instruction suivante
[F11]	Entrer dans la méthode si existante

Navigation dans le code

Raccourci	Effet
[F12]	Aller à la définition
[Shift] + [F12]	Trouver toutes les références

À vous...

Créez un projet comme décrit précédemment



Your first podcast will be awful

Your first video will be awful

Your first article will be awful

Your first art will be awful

Your first photo will be awful

Your first game will be awful

But your first code will be perfect.
Zero bugs and a very clean code.

It will be "Hello World"

Types de données simples – C#

- *sbyte, short, int, long* : Entiers signés
- *byte, ushort, uint, ulong* : Entiers non signés
- *char* : Caractères unicode
- *float, double* : Nombres à virgule flottante (IEEE)
- *decimal* : Nombres à virgule flottante à grande précision
- *bool* : Valeur booléenne, soit *true* soit *false*
- *string* : chaîne de caractères

```
int unEntier = 1;  
double unReel = 1.0;  
decimal unDecimal = 1.0m;
```

: déclaration
: initialisation

Types de données – Correspondances

Type algorithmique	Type C#	Exemples
booléen	bool	bool estAdmis = true;
entier	sbyte, short, int, long	int ecartUnitaire = 0;
N/A	byte, ushort, uint, ulong (entiers non signés)	uint nombreEdutiants = 0;
réel	float, double, decimal	float distance = 12.3f; double surfacePolygone = 23.45; decimal salaireAnnuel = 56000.45m;
caractère	char	char premiereLettreMot = 'e';
chaine	string	string salutations = "Bienvenue au cours 420-W10-SF";

Entiers et réels – Opérateurs arithmétiques

Opérateur algorithmique	Opérateur C#	Exemple	Remarque
+, -, /, *	+, -, /, *	2 + 3 // 5 2.0f + 3.2f // 5.2f 2.0m + 3.2m // 5.2m	Addition, soustraction, division et multiplication
%, /	%, /	23 % 2 // 1 23 / 2 // 11	Entiers seulement : modulo et division entière
N/A	++<variable>, --<variable>	// si i == 1 ++i // éval == 2 ; i == 2 --i // éval == 0 ; i == 0	Entiers seulement, pré-incréméntation (Pré-décréméntation) : i prend la valeur i + 1 (i - 1) et le résultat de l'opérateur est i + 1 (i - 1)
N/A	<variable>++, <variable>--	// si i == 1 i++ // éval == 1 ; i == 2 i-- // éval == 1 ; i == 0	Entiers seulement, post-incréméntation (Post-décréméntation) : i prend la valeur i + 1 (i - 1) et le résultat de l'opérateur est i

Booléen – Opérateurs logiques

Opérateur algorithmique	Opérateur C#	Exemple	Remarque
et	&&	true && false // false	Notez la répétition du caractère « & »
ou		true false // true	Notez la répétition du caractère « »
non	!	!true // false	Notez qu'il n'y a qu'un caractère « ! »

Opérateurs comparaisons – Sauf chaînes de caractères

Opérateur algorithmique	Opérateur C#	Exemple	Remarque
<, <=	<, <=	<code>13 <= 42 // true</code> <code>42 <= 13 // false</code>	
>, >=	>, >=	<code>42 > 12 // true</code> <code>13 >= 42 // false</code>	
==	==	<code>42 == 42 // true</code> <code>42 == 13 // false</code>	Notez le double égale pour différencier l'opérateur de l'affectation
!=	!=	<code>42 != 13 // true</code> <code>42 != 42 // false</code>	Non égale

Chaines de caractères – Opérateurs

Opérateur algorithmique	Opérateur C#	Exemple
[]	[]	"Bonjour"[0]. // 'B' "Bonjour"[1] // 'o'
.Longueur	.Length	"Bonjour".Length; // 7 chaine texteSaluer = "Bonjour"; texteSaluer.Length // 7
+	+	"Bonjour " + "tout le monde !" // "Bonjour tout le monde !"
N/A	.ToLower() .ToUpper()	"Bonjour".ToLower(). // "bonjour" "Bonjour".ToUpper() // "BONJOUR"

Chaines de caractères – Opérateurs

Opérateur algorithme	Opérateur C#	Exemple	
==	.Equals(<autre chaîne> (Ne pas utiliser == même si cela fonctionne en C#)	"Bonjour".Equals("Bonjour") // true "Bonjour".Equals("bonjour") // false	
!=	!<chaîne>.Equals(<autre chaîne> (Ne pas utiliser != même si cela fonctionne en C#)	!"Bonjour".Equals("Bonjour") // false !"Bonjour".Equals("bonjour") // true	
<, <=, >, >=	.CompareTo(<autre chaîne>)	"a".CompareTo("b") // -1 "a".CompareTo("a") // 0 "b".CompareTo("a") // 1	

```
"Arbre".CompareTo("arbre")); // 1 "arbre" < "Arbre"  
"Arbre".CompareTo("Banane")); // -1 "Arbre" < "Banane"  
"Banane".CompareTo("arbre")); // 1 "arbre" < "Banane"
```

"arbre" < "Arbre" < "Banane"

Les comparaisons de chaînes de caractères, en C#, suivent l'ordre lexicographique

Conversion

Les conversions d'un type vers un autre type qui a une capacité de représentation plus grande sont automatiques et compatibles.

Exemple : `int` vers `double`

Le contraire n'est pas autorisé.

De	Vers	Exemple
tous	string	<code>decimal pi = 3.1415;</code> <code>string valeurEnChaine = pi.ToString();</code>

Sorties

Fonction algorithmique	Fonction C#	Exemple
écrire(...)	Console.Out.Write(...)	Console.Out.Write("Bonjour\u00a0!"); Bonjour! █
écrireNL(...)	Console.Out.WriteLine(...)	Console.Out.WriteLine("Bonjour\u00a0!"); Bonjour! ↵ █

 \u00a0 : espace

 █ : curseur

 ↵ : retour chariot

Entrées

Lire une donnée du type	Fonction algorithmique	Fonction C#	Exemple
booléen	lire()	Console.In.ReadBool()	<code>bool estAdmis = Console.In.ReadBool();</code>
entier	lire()	Console.In.ReadInt()	<code>int nombreNotes = Console.In.ReadInt();</code>
réel	lire()	Console.In.ReadFloat() Console.In.ReadDouble() Console.In.ReadDecimal()	<code>float distance = Console.In.ReadFloat();</code> <code>double surfacePolygone = Console.In.ReadDouble();</code> <code>decimal salaireAnnuel = Console.In.ReadDecimal();</code>
char	lire()	Console.In.ReadChar()	<code>char lettre = Console.In.ReadChar();</code>
chaine	lire()	Console.In.ReadLine()	<code>string prenom = Console.In.ReadLine();</code>

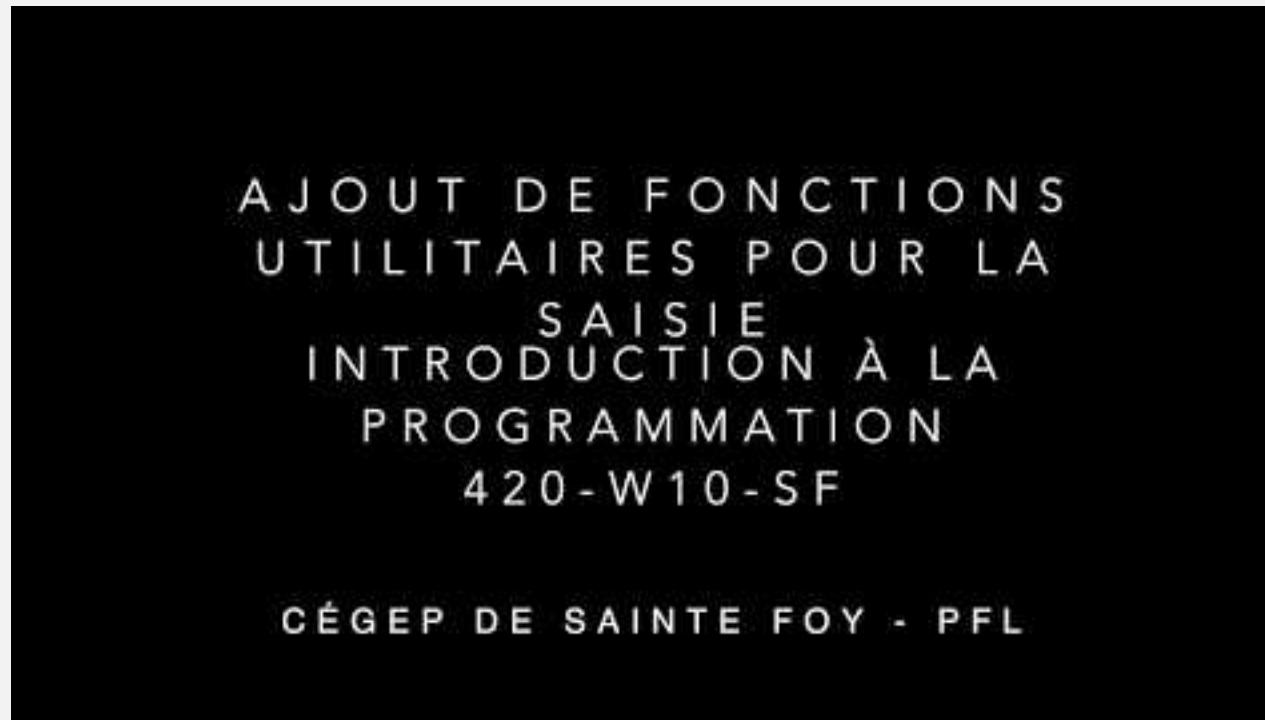
Les fonctions d'entrées C# ne sont pas standards, elles ont été écrites pour votre cours d'introduction à la programmation.
Le code est disponible sur le dépôt Git suivant : <https://gist.github.com/PiFou86/a33ca260c533298b41b2fc75da054572>
Téléchargez le fichier et déposez le dans chacun des projets Visual Studio que vous allez créer.

Nombres aléatoires

```
Random genererNombre = new Random(DateTime.Now.Millisecond);
int nombreAleatoire = 0;
[...]
// Chaque exécution de generateurAleatoire.Next(<valeurMinimale>,
// <valeurMaximaleExcluse>)
// permet d'obtenir une nouvelle valeur
nombreAleatoire = genererNombre.Next(1, 6);
```

Démo entrées / sorties

- Fonctions : <https://gist.github.com/PiFou86/a33ca260c533298b41b2fc75da054572>



<https://youtu.be/BowrzTi4Qck>

Traduction de l'algorithme du dernier cours en C#

Pseudo code

```
réel oprande1 = 0.0;  
réel oprande2 = 0.0;  
réel resultatMultiplication = 0.0;  
  
écrire("Entrez un premier réel : ");  
oprande1 = lire();  
écrire("Entrez un second réel : ");  
oprande2 = lire();  
écrireNL();  
  
resultatMultiplication = oprande1 * oprande2;  
  
écrire("La multiplication de " + oprande1.VersChaine());  
écrire(" et de " + oprande2.VersChaine() + " est : ");  
écrire(resultatMultiplication);
```

C#

```
decimal oprande1 = 0.0m;  
decimal oprande2 = 0.0m;  
decimal resultatMultiplication = 0.0m;  
  
Console.Out.Write("Entrez un premier réel : ");  
oprande1 = Console.In.ReadDecimal();  
Console.Out.Write("Entrez un second réel : ");  
oprande2 = Console.In.ReadDecimal();  
Console.Out.WriteLine();  
  
resultatMultiplication = oprande1 * oprande2;  
  
Console.Out.Write("La multiplication de " + oprande1.ToString());  
Console.Out.Write(" et de " + oprande2.ToString() + " est : ");  
Console.Out.Write(resultatMultiplication);
```

Commencer à écrire des programmes propres

- Pour le moment, vous allez créer un projet Visual Studio par feuille d'exercices
- Vous allez séparer les parties :
 - Déclaration des données
 - Lecture des données de traitement
 - Traitement
 - Affichage

Commencer à écrire des programmes propres

- Chaque exercice doit être résolu dans un sous programme propre à l'exercice et non directement dans le sous-programme « *main* »

Retenez la syntaxe donnée ici. Elle sera expliquée plus loin dans le cours.

Choisissez un nom qui a du sens pour vous

Un sous-programme débute par le caractère « { » et se terminer par le caractère « } ».

```
public static void Lire2NombresCalculerEtAfficherMultiplication()
{
    decimal operande1 = 0.0m;
    decimal operande2 = 0.0m;
    decimal resultatMultiplication = 0.0m;

    Console.Out.Write("Entrez un premier réel : ");
    operande1 = Console.In.ReadDecimal();
    Console.Out.Write("Entrez un second réel : ");
    operande2 = Console.In.ReadDecimal();
    Console.Out.WriteLine();

    resultatMultiplication = operande1 * operande2;

    Console.Out.Write("La multiplication de " + operande1.ToString());
    Console.Out.Write(" et de " + operande2.ToString() + " est : ");
    Console.Out.Write(resultatMultiplication);
}
```

On crée au moins un sous-programme par exercice.

Commencer à écrire des programmes propres

- Les appels à votre sous-programme se font à partir du sous programme « *main* »
- Vous pouvez mettre les appels inutiles en commentaires.

```
static void Main(string[] args)
{
    Lire2NombresCalculerEtAfficherMultiplication();
}
```

Existe déjà dans le fichier
« Program.cs »

N'oubliez les parenthèses

Programme complet

```
1  using System;
2
3  namespace Cours03MaPremiereApplication
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              CalculerMultiplication();
10         }
11
12         public static void CalculerMultiplication()
13         {
14             decimal operande1 = 0.0m;
15             decimal operande2 = 0.0m;
16             decimal resultatMultiplication = 0.0m;
17
18             Console.Out.WriteLine("Entrez un premier réel : ");
19             operande1 = Console.In.ReadDecimal();
20             Console.Out.WriteLine("Entrez un second réel : ");
21             operande2 = Console.In.ReadDecimal();
22             Console.Out.WriteLine();
23
24             resultatMultiplication = operande1 * operande2;
25
26             Console.Out.WriteLine("La multiplication de " + operande1.ToString());
27             Console.Out.WriteLine(" et de " + operande2.ToString() + " est : ");
28             Console.Out.WriteLine(resultatMultiplication);
29         }
30     }
31 }
```

Exercices

- Codez les exercices 3, 4, 6, 8 du cours précédent

