

INTRODUCTION AU LANGUAGE C#

UTOPIOS



ANTHONY DI PERSIO

DECOUVERTE DU C#

Caractéristiques et nouveautés du langage C#

01

LES CONVENTIONS DU C#

Tour d'horizon des spécificités du C#

02

LA CONSOLE EN C#

Instruction de lecture et écriture en mode console

03

LES TYPES DE VARIABLES

Enumération des types et variables du C#

04

TABLE DES MATIÈRES

05

LE CAST DE TYPE EN C#

Conversion entre les types : Casting et Boxing

06

LES TABLEAUX EN C#

Les opérateurs de comparaison et instructions conditionnelles

07

STRUCTURES DE CONTRÔLE

Les opérateurs de comparaison et instructions conditionnelles

08

LES MÉTHODES EN C#

Création, paramètres et retour des méthodes en C#

01

DECOUVERTE DU C#

Caractéristiques et nouveautés du langage C#

DECOUVERTE DU C#

Le **C#** (se prononce CSharp) se présente comme la suite des langages C et C++, une peu comme Java

- C'est un langage complètement orienté objet
 - Pas de possibilité d'écrire de fonction en dehors des classes, pas de variables globales
- Permet l'écriture de programme plus sûr et plus stable
 - Grâce à la gestion automatique de la mémoire à l'aide du ramasse miette (garbage collector)
 - Et la gestion des exceptions.

DECOUVERTE DU C#

- Nouveautés par rapport au C++ :
 - Libération automatique des objets
 - Disparition des pointeurs (remplacés par des références)
 - Disparition du passage d'argument par adresse au profit du passage par référence
 - Nouvelles manipulations des tableaux
 - Nouvelle manière d'écrire les boucles (foreach)
 - Disparition de l'héritage multiple mais possibilité d'implémenter plusieurs interfaces par une même classe

02

LES CONVENTIONS DU C#

Tour d'horizon des spécificités du C#

LES CONVENTIONS DU C#

- Un programme développé en C# doit respecter certaines conventions
 - Il comporte obligatoirement une fonction Main (M majuscule), c'est le point d'entrée de notre application
 - La fonction Main doit être obligatoirement membre d'une classe
 - Le point virgule à la fin de la définition d'une classe est optionnel (pas en C++)

LES CONVENTIONS DU C#

- Pour utiliser une méthode appartenant à une classe, il y a deux manières pour l'appeler
 - Spécification du nom complet
 - ✓ `System.Console.WriteLine();`
 - Spécification du nom relatif

```
using System ;  
class Prog  
{  
    static void Main()  
    {  
        Console.WriteLine("Bonjour");  
    }  
}
```


LES CONVENTIONS DU C#

- Afin de commenter le code en C#, trois type de commentaires peuvent êtres employés
 - Le commentaire le ligne
 - ✓ `//` Le reste de la ligne est commenté
 - Le commentaire multilignes
 - ✓ `/*`
Tout le texte situé entres les deux délimiteurs
`*/`
 - Le commentaire servant à la documentation automatique
 - ✓ `///` Ceci est un commentaire de documentation

LES CONVENTIONS DU C#

- Les identificateurs en C#
 - Premier caractère : **lettre** (y compris celles accentuées) ou le underscore _
 - Distinction entre minuscule et majuscule (Case Sensitive)
 - Les caractères accentués sont acceptés
 - Un mot réservé du C# peut être utilisé comme identificateur de variable à condition qu'il soit précédé de @
 - ✓ Ex. d'identificateurs : *NbLignes*, *Nbécoles*, *@int*

LES CONVENTIONS DU C#

- Les instructions en C#
 - Se terminent obligatoirement par un point virgule « ; »
 - ✓ `Console.WriteLine("Bonjour");`
- Une suite d'instructions délimité par des accolades {...} constitue un bloc
 - Les blocs définissent les zones de validité des variables (portée des variables)

LES CONVENTIONS DU C#

- Les mots réservés du C#

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true
byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace	string	

LES CONVENTIONS DU C#

- Les opérateurs du C#
 - Les opérateurs arithmétiques

Opérateur	Fonction
+	Addition
-	Soustraction
/	Division
*	Multiplication
%	Modulo (reste de la division Euclidienne)

- Les opérateurs Pré-Post Incrémentation et autres

Opérateur	Fonction
++	Incrémente de 1
--	Décrémente de 1
+=	Addition (à une variable)
-=	Soustraction (à une variable)
/=	Division (à une variable)
*=	Multiplication (à une variable)

03

LA CONSOLE EN C#

Instruction de lecture et écriture en mode console

LA CONSOLE EN C#

Opération de saisie et d'affichage en mode console

- Affichage en mode console
 - L'affichage en mode console se fait essentiellement à l'aide des méthodes
 - ✓ `Console.Write()`; //Affiche une chaine de caractères
 - ✓ `Console.WriteLine()`; //Affiche une chaine de caractères puis retourne à la ligne

LA CONSOLE EN C#

Opération de saisie et d'affichage en mode console

- Saisie en mode console
 - La saisie en mode console se fait essentiellement à l'aide des méthodes
 - ✓ `Console.Read()`; // Lit un caractère à partir du flux standard d'entrée
 - ✓ `Console.ReadLine()`; // Lit une chaîne de caractères à partir du flux standard d'entrée
 - La valeur retournée est "null" si aucune donnée n'a été saisie (l'utilisateur tape directement ENTREE)
 - `ReadLine()` ne peut retourner que des chaînes de caractères. Il faut la convertir pour les autres types

LA CONSOLE EN C#

Les caractères spéciaux dans les chaînes de caractères ([string](#))

- Comment faire pour pouvoir afficher un chemin de fichier contenant un anti slash ?

➤ C'est là qu'intervient le caractère spécial « \ »

✓ `string maChaine = "c:\\repertoire\\fichier.cs";`

```
c:\repertoire\fichier.cs  
Appuyez sur enter pour fermer le programme
```

➤ Pour ce cas particulier il est possible d'utiliser « @ »

✓ `string maChaine = @"c:\repertoire\fichier.cs";`

LA CONSOLE EN C#

Les caractères spéciaux dans les chaînes de caractères (`string`)

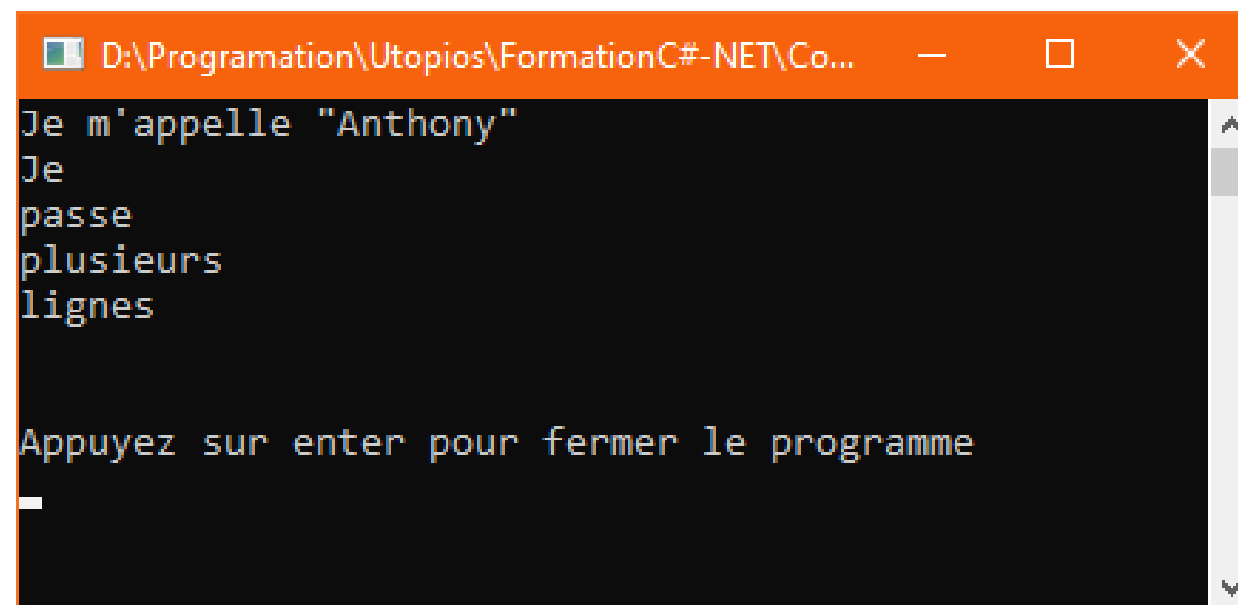
- Les chaînes de caractères étant déclarée entre ``guillemets``, comment faire pour insérer des guillemets dans les chaînes?
 - C'est là qu'intervient de nouveau le caractère spécial « \ »
 - ✓ `string maChaine = "Je m'appel \"Anthony\"";`

```
Je m'appel "Anthony"  
Appuyez sur enter pour fermer le programme
```


LA CONSOLE EN C#

Les caractères spéciaux dans les chaînes de caractères (`string`)

- Nous pouvons insérer des retours à la ligne « `\n` »
 - `string prenom = "Je m'appelle \"Anthony\"";`
 - ✓ `Console.WriteLine(prenom);`
 - ✓ `Console.WriteLine("Je\npasse\nplusieurs\nlignes\n\n");`



```
D:\Programation\Utopios\FormationC#-NET\Co...
Je m'appelle "Anthony"
Je
passe
plusieurs
lignes

Appuyez sur enter pour fermer le programme
_
```

LA CONSOLE EN C#

Les caractères spéciaux dans les chaînes de caractères ([string](#))

- Nous pouvons insérer tabulation avec « \t »
 - ✓ `Console.WriteLine(`Liste de choses à faire : `);`
 - ✓ `Console.WriteLine(`\t - Apprendre de C#`);`
 - ✓ `Console.WriteLine(`\t - Faires des exercices`);`

```
Liste de choses à faire :  
    - Apprendre le C#  
    - Faires des exercices  
Appuyez sur enter pour fermer le programme
```


04

LES TYPES DE VARIABLES

Enumération des types et variables du C#

LES TYPES ET VARIABLES

Les types de variables du C# sont de deux catégories

- Les variables de type « valeur »
 - Elle contient directement la valeur.
 - Sa durée de vie est gérée par le Garbage Collector
- Les variables de type « référence »
 - Une variable de type référence est une variable dynamique au sens du C++
 - Une telle variable référence un objet (référence mémoire vers l'objet)
 - La durée de vie de l'objet référencé par une telle variable est gérée implicitement par le Garbage Collector

LES TYPES ET VARIABLES

Comparaison de valeurs et comparaison de références

- Une variable de type "référence" contient une référence à des données stockées dans un objet. Deux variables de types références peuvent donc référencer les mêmes données
 - la modification de ces données à travers l'une de ces deux références affecte automatiquement l'autre référence
- Une variable de type "valeur" possède sa propre copie des données qu'elle stocke. Deux variables de type "valeur" peuvent avoir les mêmes données mais chacune possède sa propre copie distincte
 - La modification de l'une des deux variables n'affecte pas l'autre

LES TYPES ET VARIABLES

Les différents types de variable en C#

Type	Classe	Description	Exemples
bool	System.Bool	Booléen (vrai ou faux : true ou false)	true
char	System.Char	Caractère Unicode (16 bits) (ou plus précisément, une unité de code (code unit))	'A' 'λ' 'ω'
sbyte	System.SByte	Entier signé sur 8 bits (1 octet)	-128
byte	System.Byte	Entier non signé sur 8 bits (1 octet)	255
short	System.Int16	Entier signé sur 16 bits	-129
ushort	System.UInt16	Entier non signé sur 16 bits	1450
int	System.Int32	Entier signé sur 32 bits	-100000
uint	System.UInt32	Entier non signé sur 32 bits	8000000
long	System.Int64	Entier signé sur 64 bits	-2565018947302L
ulong	System.UInt64	Entier non signé sur 64 bits	80000000000000L
float	System.Single	Nombre à virgule flottante sur 32 bits	3.14F
double	System.Double	Nombre à virgule flottante sur 64 bits	3.14159
decimal	System.Decimal	Nombre à virgule flottante sur 128 bits	3.1415926M
string	System.String	Chaîne de caractères unicode	"C:\\windows\\system32"
object	System.Object	Tous types d'objets	Person(...){...}

LES TYPES ET VARIABLES

Les plages d'utilisation des principales variables numérique en C#

Type	Exemples
byte	Entier de 0 à 255
short	Entier de -32768 à 32767
int	Entier de -2147483648 à 2147483647
long	Entier de -9223372036854775808 à 9223372036854775807
float	Nombre simple précision de -3,402823e38 à 3,402823e38
double	Nombre double précision de -1,79769313486232e308 à 1,79769313486232e308

LES TYPES ET VARIABLES

Quelques précisions relative au C#

- En C++ une valeur **nulle** est évaluée à **false** et l'inverse à **true**
 - Ce n'est pas le cas en C# : *true* = *true* et *false* = *false*
- Le type *decimal* est utilisé pour les opérations financières
 - Il permet une très grande précision
 - Les opérations avec ce type sont deux fois plus lentes que les types *double* ou *float*

LES TYPES ET VARIABLES

La déclaration d'une variable en C#

- Les variables sont déclarées par leur **type** suivi de leur **nom**
 - `Type nomVariable;`
 - ✓ Exemple: `Int age;`
- Les variables peuvent être déclarées et initialisées en même temps
 - ✓ Exemple: `Int age=20;`

LES TYPES ET VARIABLES

La déclaration d'une variable en C#

- Toute variable doit être déclarée dans un bloc. Elle cesse d'exister en dehors de ce bloc, (pas de variable globale)
- En l'absence d'une initialisation explicite les champs d'un objet sont implicitement initialisés par le compilateur (numérique à 0 et chaîne à "")

LES TYPES ET VARIABLES

Les constantes symbolique en C#

- La déclaration des constantes symboliques se fait avec le mot-clé « *const* »
- Une constante symbolique doit être obligatoirement initialisée

➤ `const « type » « NomVariable » = « valeur »;`

✓ Exemple: `const double Pi = 3,1415926535897932;`

LES TYPES ET VARIABLES

Les chaînes de caractères (`string`) en C#

- Les chaînes de caractères en C# sont des variables de type « référence » et non de « valeur »
- Cela s'explique par le fait que lors d'une déclaration d'une chaîne de caractère (`string`), on déclare une référence à une sorte de tableau caractères (`char`) qui la contient
- On peut donc interroger « ce tableau » à un numéro de cellule
 - `string prenom = "Anthony";`
 - ✓ `Console.WriteLine(prenom[0]);` // Affichera « A »
 - `prenom` est une référence mémoire « du tableau » de type `char` [`A,n,t,h,o,n,y`]; // Effectivement à l'index [0] : « A »

05

LE CAST DE TYPE EN C#

Conversion entre les types : Casting et Boxing

LE CAST DE TYPE EN C#

Le casting d'une variable consiste à la convertir le type d'une variable en au autre type

- Nous pouvons rencontrer deux possibilités lors d'un cast de type
 - Soit les deux types sont compatibles
 - ✓ Exemple: Caster un `short` en `int`
 - Soit les deux types sont incompatibles
 - ✓ Exemple: Caster un `string` en `int`

LE CAST DE TYPE EN C#

Le casting entre deux types compatibles

- C'est le cas le plus simple de conversion de type
 - Le casting implicite (Le compilateur le fait pour nous)
 - ✓ Caster un `short` en `int` se fait implicitement

```
short @short = 200;  
int @int = @short;  
Console.WriteLine(@int); // Affichera 200
```

- ✓ En effet rentrer un petit chiffre dans un grand se fait sans efforts
- ✓ Par contre, la réciproque n'est pas vraie

LE CAST DE TYPE EN C#

Le casting entre deux types compatibles

- Caster un type potentiellement plus grand dans un plus petit
 - Il faudra employer un casting explicite
 - ✓ Caster un `int` dans un `short`

```
int @int = 200;  
short @short = (short)@int;  
Console.WriteLine(@short); // Affichera 200
```

- ✓ Attention, un casting explicite indique au compilateur que vous savez parfaitement ce que vous faites
- ✓ A grand pouvoir, grande responsabilité...

LE CAST DE TYPE EN C#

Le casting entre deux types compatibles

- Caster un type potentiellement plus grand dans un plus petit
 - L'erreur est humaine et parfois...
 - ✓ Un casting explicite mal contrôlé et c'est le bug assuré

```
int @int = 200000;  
short @short = (short)@int;  
Console.WriteLine(@short); // Affichera 3392
```

- ✓ Allez expliquer à votre client que les 196608 € manquant sont liés à une erreur de cast... Si vous la trouvez !
- ✓ A grand pouvoir, grande responsabilité...

LE CAST DE TYPE EN C#

Le casting entre deux types incompatibles

- Incompatible peut-être, mais qui ont le même sens
 - Conversion d'une chaîne de caractère `string` en `int`
 - ✓ Avec la méthode `convert()`, c'est possible...

```
string chaineAge = "20";  
int age = Convert.ToInt32(chaineAge);  
Console.WriteLine("chaineAge convertie en int : " + age);
```

- ✓ En effet « 20 » en `string` et en `int` se convertissent facilement

LE CAST DE TYPE EN C#

Le casting entre deux types incompatibles

- Conversion d'une chaîne de caractère `string` en `int`
 - Possible... mais quand la chaîne ne représente pas un entier, la conversion va échouer

```
string chaineAge = "vingt ans";  
int age = Convert.ToInt32(chaineAge);  
Console.WriteLine("chaineAge convertie en int : " + age);
```

- ✓ En effet, l'exemple ci-dessus va vous faire une erreur

```
System.FormatException : 'Input string was not in a correct format.'
```

- ✓ Le programme se retrouvera bloqué...

LE CAST DE TYPE EN C#

Résumé sur le casting entre deux types

- Il est possible, avec le casting, de convertir la valeur d'un type dans un autre lorsqu'ils sont compatibles entre eux
- Le casting explicite s'utilise en préfixant une variable par un type précisé entre parenthèses
- Le Framework **.NET** possède des méthodes permettant de convertir des types incompatibles entre eux s'ils sont sémantiquement proches

06

LES TABLEAUX EN C#

La déclaration et allocation de mémoire à un tableau

LES TABLEAUX EN C#

Déclaration d'un tableau et allocation de mémoire

- Les tableaux sont des types « **références** »
- Déclaration d'un tableau
 - `Type[] NomTab; // NomTab fait référence à un tableau`
- Allocation de l'espace mémoire d'un tableau
 - `NomTab = new Type[Taille] ; // Taille indique le Nb éléments`
- Exemples :
 - `string[] Prenoms; // Déclaration d'un tableau de string`
 - `Prénoms = new string[3] ; // Le tableau contient 3 éléments`

LES TABLEAUX EN C#

Déclaration, allocation et initialisation de valeur à un tableau

- Il est possible de déclarer et allouer l'espace en même temps
 - `Type[] NomTab = new Type[Taille] ;`
 - ✓ `string[] Prenoms = new string[3] ;`
- Une fois déclaré nous pouvons initialiser ses valeurs
 - ✓ `Prenoms = { "Tit", "Tata", "Toto" } ;`
- Il est possible de déclarer, allouer et initialiser en même temps
 - ✓ `float[] Valeur = new float[] {2.5f,0.3f,5.9f};`
 - ✓ `float[] Valeur = {2.5f,0.3f,5.9f};`

LES TABLEAUX EN C#

Tableaux avec des cellules de types différents

- Il est possible de créer des tableaux ayant des cellules de types différents
 - Le type de base de ces tableaux doit être le type *object*
- Une cellule de type *object* peut recevoir une valeur de n'importe quel type
 - `object[] Tabs = new object[3];`
 - `Tabs[0] = 12 ; Tabs[1] = 1.2 ; Tabs[2] = "Message" ;`

LES TABLEAUX EN C#

La libération d'un tableau

- La libération d'un tableau se fait automatiquement par le ramasse-miettes (Garbage Collector)
- Un tableau cesse d'exister :
 - Lorsqu'on quitte le bloc dans lequel il est déclaré
 - Lorsqu'on assigne une nouvelle zone (nouvelle valeur y compris *null*) à la référence qui désigne le tableau
 - ✓ `float[] Valeurs = {2.5f,0.3f,5.9f};`
 - ✓ `Valeurs = new float[] {3,9f,1,256f,425,68f};`
 - ✓ `Valeurs = null;`

LES TABLEAUX EN C#

La copie d'un tableau

- La copie d'un tableau est en fait une copie de sa référence
 - `int[] T1 = {2,3,4};`
 - `int[] T2; // T2 contient la valeur « null »`
 - `T2 = T1; // T1 et T2 font référence au même tableau`
- Ainsi, si vous modifiez la première valeur de T1
 - `T1[0] = 5;`
- Alors automatiquement la valeur de T2 sera
 - `T2 = {5,3,4};`

LES TABLEAUX EN C#

La copie d'un tableau

- Un autre exemple avec deux tableaux de taille différente
 - `int[] T1 = {2,3,4};`
 - `int[] T2 = new int[20];`
 - `T2 = T1;` // T1 et T2 font référence au même tableau
- T2 fait maintenant référence à la zone mémoire contenant le tableau de trois éléments
 - La zone mémoire contenant les 100 cellules est signalée "à libérer"
 - Le ramasse-miettes (Garbage Collector) la libérera lorsque un besoin en mémoire se manifestera

LES TABLEAUX EN C#

La copie d'un tableau

- Pour faire réellement une copie de tableaux il existe deux méthodes
 - La méthode **CopyTo**
 - La méthode **Clone**

LES TABLEAUX EN C#

La copie d'un tableau avec la méthode `CopyTo()`

- Utilisation de la méthode `CopyTo()`
 - `int[] T1 = {2,3,4};`
 - `int[] T2 = new int[10];` // Toutes les valeurs de T2 sont à 0
 - `T1.CopyTo(T2, 0);` // Fais la copie à partir de la cellule 0
- Maintenant nous avons bien deux tableaux distinct
 - `T1 = {2,3,4};`
 - `T2 = {2,3,4,0,0,0,0,0,0,0};`

LES TABLEAUX EN C#

La copie d'un tableau avec la méthode `Clone()`

- Utilisation de la méthode `Clone()`
 - `int[] T1 = {2,3,4};`
 - `int[] T2; // T2 = null`
 - `T2 = (int[])T1.Clone(); // Fais la copie de T1 dans T2`
 - `T1[0] = 100;`
- De nouveau, nous avons bien deux tableaux distincts
 - `T1 = {100,3,4};`
 - `T2 = {2,3,4};`

07

STRUCTURES DE CONTRÔLE

Les opérateurs de comparaison et instructions conditionnelles

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- Utiles pour des opérations qui dépendent d'un résultat précédent
 - Lors d'une opération de connexion par exemple
 - ✓ Si le login et le mot de passe sont bons, nous pouvons nous connecter
 - ✓ Sinon un message d'erreur s'affiche à l'écran
- Il s'agit de ce que l'on appelle une condition
 - Elle est évaluée lors de l'exécution et en fonction de son résultat (vrai ou faux) nous ferons telle ou telle chose

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- Les opérateurs de comparaison (Opérateurs logiques)
 - Une condition se construit grâce à des opérateurs de comparaison
 - On dénombre plusieurs opérateurs de comparaisons, les plus courants sont :

Opérateur	Description	Opérateur	Description
==	Egalité	<=	Inférieur ou égal
!=	Différence	&&	ET logique
>	Supérieur à		OU logique
<	Inférieur à	!	Négation
>=	Supérieur ou égal		

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **if** »
 - Elle permet d'exécuter du code si une condition est vraie (if = si en anglais)

```
decimal compteEnBanque = 300;  
if (compteEnBanque >= 0)  
    Console.WriteLine("Votre compte est créditeur");
```

- Si l'instruction if ne contient qu'une seule action à exécuter si elle est vraie les accolades ne sont pas obligatoire, sinon

```
decimal compteEnBanque = 300;  
if (compteEnBanque >= 0)  
{  
    Console.WriteLine("Votre compte est créditeur");  
    Console.WriteLine("Solde restant : " + compteEnBanque + " Euros");  
}
```

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **If** »
 - Il est possible de vérifier plusieurs conditions à la suite

```
decimal compteEnBanque = 300;  
if (compteEnBanque >= 0)  
    Console.WriteLine("Votre compte est créditeur");  
if (compteEnBanque < 0)  
    Console.WriteLine("Votre compte est débiteur");
```

- Une autre solution est d'utiliser le mot clé « **else** », qui veut dire « sinon » en anglais

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **else** » (sinon)
 - Elle sera toujours à la suite d'une instruction « **if** »
 - Si la valeur est vraie, alors on fait quelque chose, sinon, on fait autre chose

```
decimal compteEnBanque = 300;  
if (compteEnBanque >= 0)  
    Console.WriteLine("Votre compte est créditeur");  
else  
    Console.WriteLine("Votre compte est débiteur");
```

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **else** » (sinon)
 - Il est également possible de tester la valeur d'un booléen
 - L'exemple précédent peut aussi s'écrire

```
decimal compteEnBanque = 300;  
bool estCrediteur = (compteEnBanque >= 0);  
if (estCrediteur)  
    Console.WriteLine("Votre compte est créditeur");  
else  
    Console.WriteLine("Votre compte est débiteur");
```


STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **else** » (sinon)
 - Un type bool peut prendre deux valeurs, **vrai** ou **faux**, qui s'écrivent avec les mots clés *true* et *false*
 - Pour mieux comprendre le principe d'un booléen avec la structure conditionnelle « **if** » « **else** »

```
bool estVrai = true;
if (estVrai)
    Console.WriteLine("C'est vrai !");
else
    Console.WriteLine("C'est faux !");
```

STRUCTURES DE CONTRÔLE

Récupérer une erreur avec If...else

- Convertir avec `convert()` fait appel à la méthode `int.Parse()`
 - Quand le `int.Parse()` échoue, il provoque une erreur
 - La méthode `int.TryParse()` nous informe si la conversion s'est bien passée ou non, sans faire d'erreur

```
string chaineAge = "ABC20";
int age;
if (int.TryParse(chaineAge, out age))
{
    Console.WriteLine("La conversion est possible, age vaut " +
age);
}
else
{
    Console.WriteLine("Conversion impossible");
}
```

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **else if** » (sinon si)
 - Elle sera toujours à la suite d'une instruction « **if** » ou « **else if** »
 - Si la ou les valeurs précédentes sont fausses, alors on fait une autre comparaison.
 - Il est possible de mettre autant de « **else if** » que de conditions à vérifier

```
if (compteEnBanque > 0)
    Console.WriteLine("Votre compte est créditeur");
else if (compteEnBanque == 0)
    Console.WriteLine("Votre solde est nul");
else
    Console.WriteLine("Votre compte est débiteur");
```

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **else** » (sinon)
 - Il est également possible de combiner les tests grâce aux opérateurs de logique conditionnelle
 - Par exemple && qui correspond à l'opérateur ET

```
string login = "Jeanne";  
string motDePasse = "essai";  
if (login == "Jeanne" && motDePasse == "essai")  
    Console.WriteLine("Bienvenue Jeanne");  
else  
    Console.WriteLine("Login incorrect");
```


STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- L'instruction « **switch** »
 - L'instruction switch peut être utilisée lorsqu'une variable peut prendre beaucoup de valeurs
 - Elle permet de simplifier l'écriture, chaque valeur doit contenir un break pour sortir du switch

```
string civilite = "M.";
switch (civilite)
{
    case "M.":
        Console.WriteLine("Bonjour monsieur");
        break;
    case "Mme":
        Console.WriteLine("Bonjour madame");
        break;
    case "Mlle":
        Console.WriteLine("Bonjour mademoiselle");
        break;
}
```

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- La boucle « **while** » (« *tant que* » en Français)
 - C'est la boucle qui va nous permettre de faire quelque chose tant qu'une condition n'est pas vérifiée.
- Il y a deux possibilités d'employer la boucle « **while** »
 - Avec la vérification de la condition avant d'exécuter la boucle (« **while** »)
 - Avec la vérification de la condition après avoir exécuter une première fois la boucle (« **do** »... « **while** » => « *fais* » ... « *tant que* »)

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- La boucle « **while** » avec vérification de la condition avant l'exécution de la boucle

```
int compteur = 1;
while (compteur <= 50)
{
    Console.WriteLine("Le compteur affiche : " + compteur);
    compteur++;
}
```

- Dans cet exemple, la condition de la boucle vérifie si la condition est vraie et une fois que la boucle a affichée la valeur 50, compteur s'incrémente de 1 (compteur++) la condition devient fausse, on sort de la boucle.
- Attention aux boucles infinies, n'oubliez pas d'incrémenter ou décrémenter votre variable d'index

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- La boucle « **do... while** » qui exécute la boucle avant la vérification de la condition

```
int compteur = 1;
do
{
    Console.WriteLine("Le compteur affiche : " + compteur);
    compteur++;
} while (compteur <= 50);
```

- Le résultat semble identique qu'avec une boucle « **while** » pourtant la boucle s'est exécutée avant la vérification de la condition
- Dans ce cas essayons avec un exemple où la condition est fausse...

STRUCTURES DE CONTRÔLE

Les instructions conditionnelles

- La boucle « **do... while** » avec une condition fausse

```
int compteur = 51;
do
{
    Console.WriteLine("Le compteur affiche : " + compteur);
    compteur++;
} while (compteur <= 50);
```

- Le résultat affiche: « Le compteur affiche : 51 » et le programme se termine car après avoir exécuté une première fois la boucle, la condition a été vérifiée
- Notez que dans notre exemple après la sortie de la boucle « **do... while** » la variable compteur vaut 52.

STRUCTURES DE CONTRÔLE

Les boucles d'itération

- La boucle « **for** »
 - Elle permet de répéter une instruction tant qu'une condition est vraie
 - ✓ Dans l'exemple ci-dessous l'instruction est exécutée tant que la condition « compteur inférieur ou égal à 50 » est **vraie**

```
int compteur;  
for (compteur = 1; compteur <= 50; compteur++)  
{  
    Console.WriteLine("L'instruction a été exécutée : " + compteur + " fois");  
}
```

- ✓ Une fois le compteur arrivé à 50, le programme sort de la boucle – **Attention aux boucles infinies**

STRUCTURES DE CONTRÔLE

Les boucles d'itération

- La boucle « **for** »
 - Elle peut être utilisée pour énumérer le contenu d'un tableau et afficher son contenu
 - ✓ Dans l'exemple ci-dessous la variable d'itération « **i** » est directement déclarée dans la boucle **for**
 - ✓ Le nombre de fois ou la boucle est exécutée est conditionné par la longueur du tableau lui-même

```
string[] joursSem = new string[] { "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche" };  
for (int i = 0; i < joursSem.Length; i++)  
{  
    Console.WriteLine(joursSem[i]);  
}
```

STRUCTURES DE CONTRÔLE

Les boucles d'itération

- La boucle « **foreach** » (Pour chaque... en Français)
 - C'est un opérateur spécialement conçu pour parcourir des listes et tableaux
 - ✓ Pour cela la variable d'itération est implicite et s'adapte automatiquement à la longueur du tableau ou de la liste
 - ✓ Dans l'exemple, noter la déclaration d'une variable temporaire « **jour** » dans laquelle sera stocké le contenu de chaque cellule pour chaque itération

```
string[] jourSem = new string[] { "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche" };  
foreach (string jour in jourSem)  
{  
    Console.WriteLine(jour);  
}
```

STRUCTURES DE CONTRÔLE

Les boucles d'itération

- La boucle « **foreach** »
 - Attention, la boucle « **foreach** » est une boucle en **lecture seule**. Cela veut dire qu'il n'est pas possible de modifier l'élément de l'itération en cours
 - ✓ L'exemple ci-dessous provoquera une erreur de compilation ; Il est impossible d'assigner à 'jour', une valeur car il s'agit d'une 'variable d'itération'

```
List<string> jours = new List<string> { "Lundi", "Mardi", "Mercredi",  
    "Jeudi", "Vendredi", "Samedi", "Dimanche" };  
foreach (string jour in jours)  
{  
    jour = "pas de jour !";  
}
```

(variable locale) string jour

Impossible d'assigner à 'jour', car il s'agit d'un 'variable d'itération foreach'

STRUCTURES DE CONTRÔLE

Les boucles d'itération

- Pour modifier le contenu d'une liste ou d'un tableau
 - Il faudra passer par une boucle **for**
 - ✓ L'exemple ci-dessous permet de modifier le contenu de la liste en assignant à chaque cellule la chaîne « pas de jour ! » en lieu et place des valeurs présentes

```
List<string> jourSem = new List<string> { "Lundi", "Mardi", "Mercredi",  
    "Jeudi", "Vendredi", "Samedi", "Dimanche" };  
for (int i = 0; i < jourSem.Count; i++)  
{  
    jourSem[i] = "pas de jour !";  
}
```

STRUCTURES DE CONTRÔLE

Les boucles d'itération

- Pour supprimer une valeur contenue dans une liste

➤ Précisions avec l'emploi d'une boucle **for**

```
for (int i = 0; i < jourSem.Count; i++)  
{  
    if (jourSem[i] == "Mercredi")  
        jourSem.Remove(jourSem[i]);  
}
```

- Comme nous supprimons un élément de la liste, nous allons nous retrouver avec une incohérence entre l'indice en cours et l'élément que nous essayons d'atteindre.
- Lorsque le jour courant est Mercredi, l'indice *i* vaut 2. Si l'on supprime cet élément, c'est Jeudi qui va prendre la position 2. Et donc rater son analyse car la boucle va continuer à l'indice 3, qui sera Vendredi

STRUCTURES DE CONTRÔLE

Les boucles d'itération

- Pour supprimer une valeur contenue dans une liste
 - On peut éviter ce problème en parcourant la boucle à l'envers

```
for (int i = jourSem.Count - 1; i >= 0; i--)  
{  
    if (jourSem[i] == "Mercredi")  
        jourSem.Remove(jourSem[i]);  
}
```

- Ainsi, la valeur qui prendra remplacera la valeur supprimée sera une valeur déjà traitée et la décrémentation de l'indice permettra de continuer avec les autres valeur à tester

STRUCTURES DE CONTRÔLE

Les instructions « **break** » et « **continue** »

- Il est possible de sortir prématurément d'une boucle grâce à l'instruction « **break** »
 - Dès qu'elle est rencontrée, elle sort du bloc de code de la boucle
 - L'exécution du programme continue alors avec les instructions situées après la boucle

```
string[] jours = new string[] { "Lundi", "Mardi",  
    "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche" };  
foreach (string jour in jours)  
{  
    Console.WriteLine("Je cherche...");  
    if (jour == "Jeudi")  
    {  
        Console.WriteLine("Trouvé !");  
        break;  
    }  
}
```

STRUCTURES DE CONTRÔLE

Les instructions « **break** » et « **continue** »

- Il est également possible de passer à l'itération suivante d'une boucle grâce à l'utilisation du mot-clé « **continue** »
 - Dès qu'elle est rencontrée, elle passe à l'itération suivante sans exécuter le reste des instructions de la boucle

```
for (int i = 1; i <= 20; i++)  
{  
    if (i % 2 != 0)  
    {  
        continue;  
    }  
    Console.WriteLine(i);  
}
```


08

LES MÉTHODES EN C#

Création, paramètres et retour des méthodes en C#

LES MÉTHODES EN C#

- Une méthode regroupe un ensemble d'instructions, pouvant prendre des paramètres et pouvant renvoyer une valeur
 - On parle parfois de « fonction » ou même « procédure » à la place du mot « méthode », cela signifie la même chose
- Le but d'une méthode est de factoriser du code afin d'éviter d'avoir à le répéter
 - Cela fait moins de travail... et c'est pas rien :)
 - Si nous devons apporter une modification a ce bout de code et s'il est dupliqué à plusieurs endroits, cela augmente davantage le travail (et évite les oublis)

LES MÉTHODES EN C#

- Ce souci de factorisation est connu comme le principe « **DRY** » (Dont Repeat Yourself!)
 - Le but de ce principe est de ne jamais (à quelques exceptions près bien sûr...) avoir à réécrire la même ligne de code
- Pour comprendre le principe des méthodes commençons par écrire quelques instruction afin de souhaiter la bienvenue à un nouvel utilisateur

```
Console.WriteLine("Bonjour à toi !");  
Console.WriteLine("-----" + Environment.NewLine);  
Console.WriteLine("\tBienvenue dans le monde merveilleux du C#" + Environment.NewLine);  
Console.WriteLine("-----" + Environment.NewLine);
```

LES MÉTHODES EN C#

- Si plus tard, on veut réafficher le message de bienvenue, il faudra réécrire ces 4 lignes de codes
 - Sauf si nous utilisons une méthode

```
static void AffichageBienvenue()  
{  
    Console.WriteLine("Bonjour à toi !");  
    Console.WriteLine("-----" + Environment.NewLine);  
    Console.WriteLine("\tBienvenue dans le monde merveilleux du C#" + Environment.NewLine);  
    Console.WriteLine("-----" + Environment.NewLine);  
}
```

- Cette méthode s'appelle AffichageBienvenue()
- Elle contient nos 4 lignes de code écrites précédemment, entre accolades, c'est « le corps » de la méthode

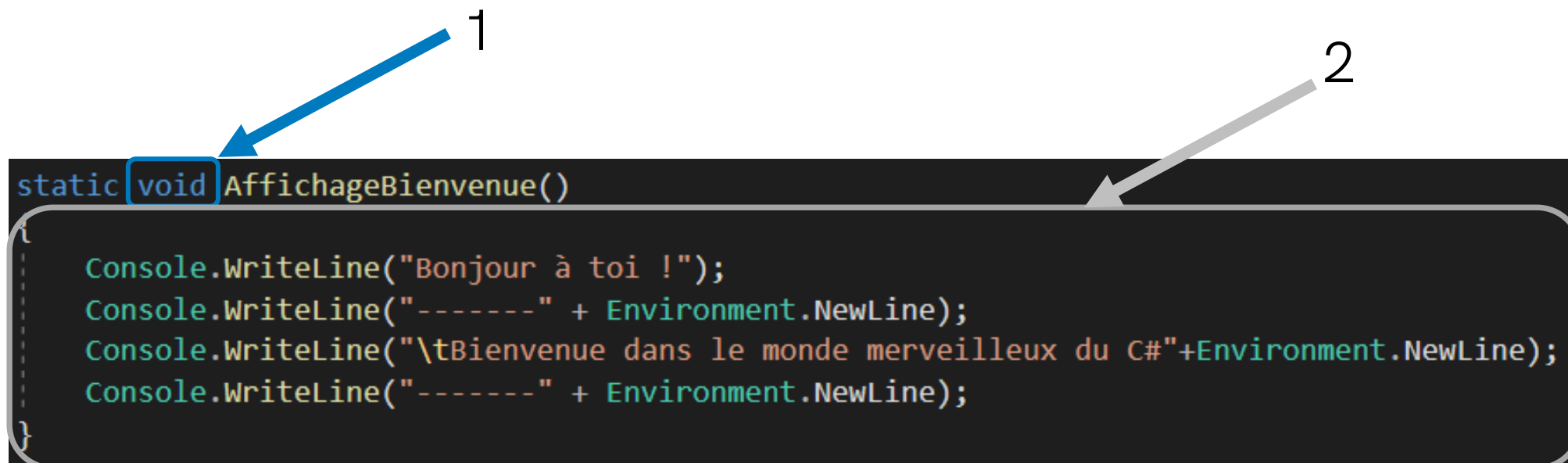
LES MÉTHODES EN C#

- Etudions notre première méthode
 - Commençons par la première ligne

```
static void AffichageBienvenue()
```
 - Son type est ce qu'on appelle la signature de la méthode (ici void)
 - ✓ Elle nous renseigne sur les paramètres de la méthode et sur ce qu'elle va renvoyer
 - Le mot clé `void` signifie que la méthode ne renvoie rien
 - Les parenthèses vides à la fin de la signature indiquent que la méthode n'a pas de paramètres
 - Le mot clé `static` ne nous intéresse pas pour l'instant, mais dans ce contexte il est obligatoire.

LES MÉTHODES EN C#

- En résumé, pour déclarer une méthode, nous aurons
 - la signature de la méthode (1)
 - Le corps de la méthode (2) contenant le bloc de code



```
static void AffichageBienvenue()  
{  
    Console.WriteLine("Bonjour à toi !");  
    Console.WriteLine("-----" + Environment.NewLine);  
    Console.WriteLine("\tBienvenue dans le monde merveilleux du C#" + Environment.NewLine);  
    Console.WriteLine("-----" + Environment.NewLine);  
}
```

LES MÉTHODES EN C#

- Nous pouvons désormais « **appeler** » la méthode (l'exécuter)

```
0 références
static void Main(string[] args)
{
    AffichageBienvenue();

    Console.WriteLine("Appuyez sur enter pour fermer le programme");
    Console.Read();
}

1 référence
static void AffichageBienvenue()
{
    Console.WriteLine("Bonjour à toi !");
    Console.WriteLine("-----" + Environment.NewLine);
    Console.WriteLine("\tBienvenue dans le monde merveilleux du C#" + Environment.NewLine);
    Console.WriteLine("-----" + Environment.NewLine);
}
```

LES MÉTHODES EN C#

- La méthode spéciale `Main()`
 - C'est le point d'entrée de notre application
 - ✓ Au démarrage, le `CLR` recherche cette méthode et commence par exécuter les instructions qu'elle contient
 - Elle doit être accessible en tout point de l'application
 - ✓ C'est pourquoi elle sera toujours « `static` »
 - Sa signature contient un certain nombre de choses entre parenthèse (`string[] args`)
 - ✓ Elle contient des paramètres, nous y reviendrons...

LES MÉTHODES EN C#

- Les paramètres d'une méthode (Surcharge)
 - Il permettent d'augmenter les possibilités de réemploie d'une méthode
 - Il se situent entre les parenthèses après le nom
 - ✓ Dans l'exemple ci-dessous les **paramètres prénom** et **langage** permettent de personnaliser le message de bienvenue

```
static void AffichageBienvenue(string prenom, string langage)
{
    Console.WriteLine("Bonjour " + prenom + " !");
    Console.WriteLine("-----" + Environment.NewLine);
    Console.WriteLine("\tBienvenue dans le monde merveilleux du " + langage );
    Console.WriteLine("-----" + Environment.NewLine);
}
```

LES MÉTHODES EN C#

- Les paramètres d'une méthode
 - Nous pouvons désormais appeler cette méthode et passer plusieurs utilisateurs en paramètres

```
static void Main(string[] args)
{
    AffichageBienvenue("Anthony", "C#");
    AffichageBienvenue("Jeanne", "Javascript");
}
```

- ✓ L'appel de la méthode respecte bien le contrat (ses paramètres) puisque nous lui passons bien deux chaînes de caractères en paramètres : le prénom et le langage.

LES MÉTHODES EN C#

- Les paramètres d'une méthode par référence
 - Les paramètres déclarés pour une méthode sans in, ref ou out sont passés à la méthode appelée par valeur.
 - Cette valeur peut être modifiée dans la méthode, mais la valeur initiale n'est pas conservée quand le contrôle repasse à la procédure appelante.
 - En utilisant un mot clé de paramètre de méthode, vous pouvez modifier ce comportement.

LES MÉTHODES EN C#

Cette section décrit les mots clés que vous pouvez utiliser pour déclarer des paramètres de méthode :

- params spécifie que ce paramètre peut prendre un nombre variable d'arguments.
- in spécifie que ce paramètre est passé par référence, mais qu'il est lu uniquement par la méthode appelée
- ref spécifie que ce paramètre est passé par référence et qu'il peut être lu ou écrit par la méthode appelée
- out spécifie que ce paramètre est passé par référence et qu'il est écrit par la méthode appelée

LES MÉTHODES EN C#

- Le retour d'une méthode
 - Une méthode peut aussi renvoyer une valeur, elle effectue un « return » (retour en Anglais)

```
static double Additionner(double nombreUn, double nombreDeux)
{
    double sommeDesNombres = nombreUn + nombreDeux;
    return sommeDesNombres;
}
```

- ✓ Ici, la méthode prend deux paramètres: deux nombres (double)
- ✓ Dans le corps, une instruction les additionne
- ✓ Puis nous retourne le résultat final: sommeDesNombres

LES MÉTHODES EN C#

Les méthodes en résumé

- Une méthode regroupe un ensemble d'instructions pouvant prendre des paramètres et pouvant renvoyer une valeur
- Les paramètres d'une méthode doivent être utilisés avec le bon type
- Le point d'entrée d'un programme est la méthode statique `Main()`
- Une méthode qui ne renvoie rien est préfixée du mot-clé `void`
- Le mot-clé `return` permet de renvoyer une valeur du type de retour de la méthode, à l'appelant de cette méthode