1

Éléments du langage

Hello world !

```
using System;

public class MaClasse
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello world !");
    }
}
```

Ce code affiche sur la console « Hello world !». La première ligne permet d'utiliser toutes les classes contenues dans l'espace de noms `System` du .NET Framework. La deuxième ligne permet de définir une classe contenant des variables et des méthodes.

Dans cet exemple, la classe `MaClasse` contient une méthode statique `Main()` qui représente le point d'entrée de toute application console .NET, c'est-à-dire que cette méthode sera appelée automatiquement lors du lancement du programme.

Les commentaires

```
// Un commentaire
```

```
/* Un commentaire sur
   plusieurs lignes */
```

```
/// <summary>
/// Commentaire pour documenter un identificateur
/// </summary>
```

Les commentaires sont des lignes de code qui sont ignorées par le compilateur et permettent de documenter votre code.

Les commentaires peuvent être :

- entourés d'un slash suivi d'un astérisque `/*` et d'un astérisque suivi d'un slash `*/`. Cela permet d'écrire un commentaire sur plusieurs lignes ;
- placés après un double slash `//` jusqu'à la fin de la ligne.

Les commentaires précédés par un triple slash sont des commentaires XML qui permettent de documenter des identificateurs tels qu'une classe ou une méthode. Le compilateur récupère ces commentaires et les place dans un document XML qu'il sera possible de traiter afin de générer une documentation dans un format particulier (HTML, par exemple).

Les identificateurs

Les identificateurs permettent d'associer un nom à une donnée. Ces noms doivent respecter certaines règles édictées par le langage.

- Tous les caractères alphanumériques Unicode UTF-16 sont autorisés (y compris les caractères accentués).
- Le souligné _ est le seul caractère non alphanumérique autorisé.
- Un identificateur doit commencer par une lettre ou le caractère souligné.
- Les identificateurs respectent la casse ; ainsi `mon_identificateur` est différent de `MON_IDENTIFICATEUR`.

Voici des exemples d'identificateurs :

```
identificateur       // Correct
IDENTificateur       // Correct
5Identificateurs     // Incorrect : commence par un
                     // chiffre
identificateur5      // Correct
_mon_identificateur  // Correct
mon_identificateur   // Correct
mon  identificateur  // Incorrect : contient un
                     // espace
*mon-identificateur  // Incorrect : contient des
                     // caractères incorrects
```

Les identificateurs ne doivent pas correspondre à certains mots-clés du langage C#, dont le Tableau 1.1 donne la liste.

Tableau 1.1 : Liste des noms d'identificateur non autorisés

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

Les variables

```
// Déclarer une variable
<type> <nomVariable>;
```

```
// Affecter une valeur à une variable
<nomVariable> = <uneValeur>;
```

Une variable est un emplacement mémoire contenant une donnée et nommé à l'aide d'un identificateur. Chaque variable doit être d'un type préalablement défini *qui ne peut changer au cours du temps*.

Pour créer une variable, il faut d'abord la déclarer. La déclaration consiste à définir le type et le nom de la variable.

```
int unEntier; // Déclaration d'une variable nommée
              // unEntier et de type int
```

On utilise l'opérateur d'affectation = pour affecter une valeur à une variable. Pour utiliser cet opérateur, il faut que le type de la partie gauche et le type de la partie droite de l'opérateur *soient les mêmes*.

```
int unEntier;
int autreEntier;
double unReel;
// Affectation de la valeur 10 à la variable unEntier
unEntier = 10;

// Affectation de la valeur de la variable unEntier
// dans autreEntier
autreEntier = unEntier

// Erreur de compilation : les types ne sont pas
// identiques des deux côtés de l'opérateur =
autreEntier = unReel
```

L'identificateur d'une variable doit être unique dans une portée d'accolades ouvrante { et fermante }.

```
{
  int entier1;       // Correct
  ...
  int entier1;       // Incorrect
  {
     int entier1;    // Incorrect
  }
}
```

Déclarer une variable avec *var* (C# 3.0)

```
// Déclarer une variable avec var
var <nomVariable> = <valeur>;
```

Le mot-clé `var` permet de déclarer une variable typée. Le type est déterminé automatiquement par le compilateur grâce au type de la valeur qui lui est affecté. L'affectation doit forcément avoir lieu au moment de la déclaration de la variable :

```
var monEntier = 10;
```

Étant donné que cette variable est typée, le compilateur vérifie si l'utilisation de cette dernière est correcte. L'exemple suivant illustre cette vérification.

```
var monEntier = 10;

monEntier = 1664;  // Correct
monEntier = 'c';   // Erreur de compilation car
                   // monEntier est de type int
```

Attention

Évitez d'utiliser le mot-clé `var` car cela rend le code plus difficile à comprendre ; il est en effet plus difficile de connaître immédiatement le type d'une variable.

Les types primitifs

Le langage C# inclut des types primitifs qui permettent de représenter des données informatiques de base (c'est-à-dire les nombres et les caractères). Le programmeur devra

utiliser ces types de base afin de créer de nouveaux types plus complexes à l'aide des classes ou des structures.

Les types primitifs offerts par C# sont listés au Tableau 1.2.

Tableau 1.2 : Les types primitifs de C#

Type	Portée	Description
`bool`	`true` or `false`	Booléen 8 bits
`sbyte`	–128 à 127	Entier 8 bits signé
`byte`	0 à 255	Entier 8 bits non signé
`char`	U+0000 à U+ffff	Caractère Unicode 16 bits
`short`	–32 768 à 32 767	Entier 16 bits signé
`ushort`	0 à 65 535	Entier 16 bits non signé
`int`	-2^{31} à $2^{31}-1$	Entier 32 bits signé
`uint`	0 à $2^{32}-1$	Entier 32 bits non signé
`float`	$\pm 1{,}5e^{-45}$ à $\pm 3{,}4e^{38}$	Réel 32 bits signé (virgule flottante)
`long`	-2^{63} à $2^{63}-1$	Entier 64 bits signé
`ulong`	0 à $2^{64}-1$	Entier 64 bits signé
`double`	$\pm 5{,}0e^{-324}$ à $\pm 1{,}7e^{308}$	Réel 64 bits signé (virgule flottante)
`decimal`	$\pm 1{,}0e^{-28}$ à $\pm 7{,}9e^{28}$	Réel 128 bits signé (grande précision)

Le choix d'un type de variable dépend de la valeur qui sera contenue dans celle-ci. Il faut éviter d'utiliser des types occupant beaucoup de place mémoire pour représenter des données dont les valeurs sont très petites. Par exemple, si l'on veut créer une variable stockant l'âge d'un être humain, une variable de type `byte` suffit amplement.

Les constantes

```
// Déclarer une constante nommée
const <type> <nomConstante> = <valeur>
```

```
'A'             // Lettre majuscule A
'a'             // Lettre minuscule a
10              // Entier 10
0x0A            // Entier 10 (exprimé en hexadécimale)
10U             // Entier 10 de type uint
10L             // Entier 10 de type long
10UL            // Entier 10 de type ulong
30.51           // Réel 30.51 de type double
3.51e1          // Réel 30.51 de type double
30.51F          // Réel 30.51 de type float
30.51M          // Réel 30.51 de type decimal
```

En C#, il existe deux catégories de constantes : les constantes non nommées qui possèdent un type et une valeur et les constantes nommées qui possèdent en plus un identificateur.

Lors de l'affectation d'une constante à une variable, le type de la constante et celui de la variable doivent correspondre.

```
long entier;
entier = 10L;        // Correct : la constante est
                     // de type long
entier = 30.51M ;    // Incorrect : la constante est
                     // de type decimal
```

Une constante nommée se déclare presque comme une variable, excepté qu'il faut obligatoirement l'initialiser avec une valeur au moment de sa déclaration. Une fois déclarée, il n'est plus possible de modifier la valeur d'une constante.

```
const double pi = 3.14159;
const double constante;    // Incorrect : doit être
                           // initialisé
double périmètre;
périmètre = pi * 20;
pi = 9.2;                   // Incorrect : il est
                            // impossible de changer
                            // la valeur d'une constante
```

Les tests et conditions

```
if (<condition>)
{
   // Code exécuté si condition est vrai
}
[else if (<autreCondition>)
{
   // Code exécuté si autreCondition est vraie
}]
[else
{
   // Code exécuté si condition et autreCondition
   // sont fausses
}]
```

```
<résultat> = <test> ? <valeur si vrai> :
                     ➥<valeur si faux>
```

```
switch (<uneValeur>)
{
   case <val1>:
      // Code exécuté si uneValeur est égale à val1
      break;

   case <val2>:
```

```
   case <val3>:
      // Code exécuté si uneValeur est égale à
      // val2 ou val3
      break;

   [default:
      // Code exécuté si uneValeur est différente
      // de val1, val2 et val3
      break;]
}
```

L'instruction `if` permet d'exécuter des instructions uniquement si la condition qui la suit est vraie. Si la condition est fausse alors, les instructions contenues dans le bloc `else` sont exécutées. Le bloc `else` est facultatif ; en l'absence d'un tel bloc, si la condition spécifiée dans le `if` est fausse, aucune instruction ne sera exécutée.

La condition contenue dans le `if` doit être de type booléen.

L'exemple suivant affiche des messages différents en fonction d'un âge contenu dans une variable de type `int`.

```
if (âge <= 50)
{
    Console.WriteLine("Vous êtes jeune !");
}
else
{
    Console.WriteLine("Vous êtes vieux ;-) !");
}
```

Il existe une variante condensée du `if` qui utilise les symboles (`?`) et (`:`). Elle permet en une seule ligne de retourner un résultat en fonction d'une condition. L'exemple qui suit illustre cette variante en retournant `false` si la valeur contenue dans `âge` est inférieure à 50 ou `true` dans le cas contraire.

```
bool vieux;
vieux = âge <= 50 ? false : true;
```

L'instruction `switch` permet de tester une valeur spécifiée par rapport à d'autres valeurs. Si l'une des valeurs correspond à la valeur testée, alors le code associé est automatiquement exécuté. Si aucune valeur ne correspond à la valeur testée, alors le code associé à clause `default` (si elle existe) sera exécuté.

Attention

Veillez à ne pas oublier l'instruction `break` entre chaque `case`, sinon les instructions associées aux valeurs suivantes seront exécutées.

Le `switch` ne peut être utilisé qu'avec les types entiers, `char`, `bool` ainsi que les énumérations et les chaînes de caractères.

L'exemple suivant affiche des messages différents en fonction du sexe d'une personne contenu dans une variable de type `char`.

```
switch (sexe)
{
   case 'M':
      Console.WriteLine("Vous êtes un homme !");
      break;

   case 'F':
      Console.WriteLine("Vous êtes une femme !");
      break;

   default:
      Console.WriteLine("Vous êtes un extraterrestre
      ➥ !");
      break;
}
```