Introduction

Vous trouverez ici un récapitulatif des mots-clés de C# traités dans cet ouvrage, triés par ordre alphabétique. Chacun est accompagné d'un rapide descriptif et d'un exemple simple d'utilisation.

abstract

Utilisé avec une classe, définit que la classe ne peut pas être instanciée.

Utilisé avec une fonction/propriété, il indique que son implémentation devra être fournie par un type dérivé.

```
public abstract class Automobile
{
    public abstract void Demarrer();
}
```

as

Opérateur de transtypage. Lorsqu'une variable ne peut pas être transtypée, il renvoie `null`.

```
string maChaine = monObjet as string;
```

ascending

Définit un tri par ordre croissant lorsqu'il est utilisé avec le mot-clé `orderby` dans une requête LINQ.

```
var requete = from c in ListeClients
              orderby c.Nom ascending
              select c;
```

async et await

`async` marque une méthode comme pouvant être exécutée de manière asynchrone. Le type de retour des méthodes asynchrones doit être `void`, `Task` ou `Task<T>`.

`await` permet d'attendre la fin de l'exécution d'une tâche asynchrone avant d'exécuter la suite du code de la méthode.

```
public async Task<string> ExecuterAsync()
{
    return await Task.Run(() => "Coucou !");
}
```

base

Fournit un accès à un membre de la classe de base.

```
public class MonType : Object
{
    public string Afficher()
    {
           string texte = base.ToString();
           Console.WriteLine(texte);
    }
}
```

bool

Alias C# du type de données `System.Boolean`. Une variable de ce type a pour valeur `true` ou `false`.

```
bool estAffiche = true;
```

break

Dans un bloc `switch`, doit être présent à la fin de chaque étiquette `case` dans laquelle aucune instruction `return` n'est présente.

```
switch (valeur)
{
    case 0:
          Console.WriteLine("Cas 0");
          break;
}
```

Dans une boucle, permet la sortie de cette boucle.

```
int i = 0;
while (true)
{
    if (i == 7)
          break;
    i++
}
```

byte

Alias C# du type `System.Byte`. Une variable de ce type contient un numérique entier compris entre 0 et 255.

```
byte number = 42;
```

case

Associe un bloc d'instruction à une valeur numérique ou d'énumération dans un bloc `switch`.

```
switch (valeur)
{
    case 0:
          Console.WriteLine("Cas 0");
          break;
}
```

catch

Définit un bloc de gestion d'exception à la suite d'un bloc `try`.

```
try
{
    System.IO.File.Open("C:\\fichier.txt",
System.IO.FileMode.Open);
}
catch (System.IO.FileNotFoundException)
{
   //Gestion des exceptions de type FileNotFoundException
}
catch (Exception)
{
   //Gestion de toutes les exceptions non gérées
}
```

char

Alias C# du type `System.Char`. Contient un caractère Unicode défini par sa forme textuelle ou numérique.

```
char lettreC = 'C';
char lettreCNum =  '\u0063';
```

class

Définit une classe.

```
class MonType
{
}
```

const

Marque une variable comme constante. Les constantes sont des types particuliers de variables statiques, donc accessibles à travers le nom du type auquel elles appartiennent.

```
public const string NomFichierConfiguration = "config.xml";
```

continue

Permet d'interrompre l'exécution d'une itération d'une boucle pour passer à l'itération suivante.

```
for (int i = 0; i < 100; i++)
{
    if (i % 2 == 0)
        continue;
}
```

decimal

Alias C# du type `System.Decimal`. Une variable de ce type contient un numérique décimal compris entre ±1,0e-28 et ±7,9e28.

```
decimal valeurDecimal = 42.000042;
```

default

Utilisé comme étiquette dans un bloc `switch`, il permet d'exécuter une portion de code si aucune autre étiquette ne correspond à la valeur testée.

```
switch (valeur)
{
    case 0:
          Console.WriteLine("Cas 0");
          break;
    case 1:
          Console.WriteLine("Cas 1");
          break;
    default:
          Console.WriteLine("Ni le cas 0, ni le cas 1");
          break;

}
```

Il peut également être utilisé pour obtenir la valeur par défaut pour un type particulier.

```
int valeurDefautInt = default(int);
```

delegate

Type représentant une signature de fonction. Les délégués sont notamment utilisés pour fixer la signature des gestionnaires d'événements.

```
public delegate void EventHandler(object sender, EventArgs args);
```

descending

Définit un tri par ordre décroissant lorsqu'il est utilisé avec le mot-clé `orderby` dans une requête LINQ.

```
var requete = from c in ListeClients
              orderby c.Nom descending
              select c;
```

do

Utilisé pour débuter une structure de boucle `do ... while`. Ce type de boucle est exécuté au minimum une fois avant l'évaluation de la condition de sortie.

```
do
{
   Console.WriteLine(i);
} while (i < 5);
```

double

Alias C# du type `System.Double`. Une variable de ce type contient un numérique décimal compris entre ±5,0e-324 et ±1,7e308.

```
double valeurDecimal = 42.000042;
```

dynamic

Permet d'utiliser des variables dont le type n'est connu qu'au moment de l'exécution.

```
dynamic variableDynamic = "une valeur";
```

else

Utilisé dans la structure d'évaluation de condition `if ... else if ... else`. Il définit un bloc gérant les cas non vérifiés par la (les) condition(s) précédente(s).

```
if (valeur == 5)
{
}
else if (valeur > 5)
{
}
else
{
}
```

enum

Définit une énumération.

```
public enum TypePersonne
{
    Prospect,
    Client
}
```

event

Définit un événement qui peut être généré par sa classe parente.

```
public event EventHandler ExecutionTerminee;
```

false

Une des deux valeurs pour le type `System.Boolean`.

```
bool estActif = false;
```

finally

Utilisé dans l'écriture de structure `try ... catch ... finally`. Définit un bloc de code exécuté même si le blog `try` génère une exception.

```
try
{
}
finally
```

```
{
}
```

float

Alias C# du type `System.Single`. Une variable de ce type contient un numérique décimal compris entre ±1,5e-45et ±3,4e38.

```
float valeurDecimal = 42.000042;
```

for

Définit une boucle pour laquelle sont spécifiées une instruction d'initialisation, une condition de sortie et une instruction d'itération. Son nombre d'itérations est généralement connu avant son exécution.

```
for (int i = 0; i < 5; i++)
{
}
```

foreach

Définit une boucle parcourant tous les éléments d'une collection implémentant l'interface `IEnumerable` ou sa version générique `IEnumerable<T>`.

```
string[] tableau = new string[10];
foreach (string element in tableau)
{
}
```

from

Définit une source de données utilisée dans une requête LINQ.

```
var requete = from c in ListeClients
              select c;
```

get

Définit un bloc de code exécuté à la lecture d'une propriété. Si ce bloc n'est pas défini, la propriété est en écriture seule.

```
public string Message
{
```

```
    get { return "Un message"; }
}
```

goto

Définit un saut de l'exécution du code vers une étiquette.

```
Console.WriteLine("Avant le goto");
goto monEtiquette;
Console.WriteLine("Après le goto : jamais executé ");
monEtiquette:
Console.WriteLine("après l'étiquette ");
```

if

Définit un bloc de code exécuté uniquement si l'évaluation de la condition associée renvoie `true`.

```
if (i == 5)
{
}
```

in

Utilisé dans la définition d'une structure `foreach` ou dans une requête LINQ pour indiquer la collection source d'une variable.

```
foreach (string element in tableau)
{
}
```

```
var requete = from client in context.Clients
              select client.Nom;
```

int

Alias C# du type `System.Int32`. Une variable de ce type contient un numérique entier entre -2 147 483 648 et 2 147 483 647.

```
int nombreOiseaux = 123;
```

interface

Définit une interface. Le nom d'une interface commence, par convention, par un i majuscule.

```
public interface ITestable
{
}
```

internal

Modificateur de visibilité permettant de rendre un élément accessible par tous les types définis dans le même assembly.

```
internal class ClasseInterne
{
}
```

is

Opérateur de comparaison de type. Renvoie `true` si la variable indiquée est du type spécifié.

```
object valeur = "Ceci est une chaîne de caractères";
bool estUneChaine = valeur is string;
```

join

Permet d'effectuer une jointure entre deux collections de données dans une requête LINQ.

```
var requete =
    from c in ListeClients
    join comm in ListeCommandes on c.IdClient = comm.IdClient
    select new { c.IdClient, comm.IdCommande };
```

long

Alias C# du type `System.Int64`. Une variable de ce type contient un numérique entier compris entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807.

```
long nombreOiseaux = 123;
```

namespace

Définit un espace de noms.

```
namespace MonApplication.ViewModel
{
}
```

new

Opérateur d'instanciation.

```
Client client = new Client();
```

Peut également être utilisé pour définir une contrainte sur un élément générique. Dans ce contexte, il force un type paramètre à avoir un constructeur sans paramètre.

```
public class Liste<T> where T : new ()
{
}
```

null

Valeur spéciale indiquant qu'un objet de type référence ne possède aucune valeur.

```
Client client = null;
if (client == null)
    client = new Client();
```

object

Alias C# du type `System.Object`. C'est la classe de base de tous les types .NET.

```
object obj = "Chaîne de caractères enregistrée dans une variable
de type object";
```

operator

Permet de redéfinir un opérateur standard de C#.

```
public static CollectionDeTimbres operator +(CollectionDeTimbres
collection1, CollectionDeTimbres collection2)
{
   CollectionDeTimbres resultat;
   resultat.nombreDeTimbres = collection1.nombreDeTimbres +
collection2.nombreDeTimbres;
   resultat.prixEstime = collection1.prixEstime +
```

```
collection2.prixEstime;

   return resultat;
}
```

orderby

Définit que les résultats d'une requête LINQ doivent être triés selon une ou plusieurs colonnes.

```
var requete = from c in ListeClients
              orderby c.Nom descending
              select c;
```

out

Définit un paramètre de méthode comme étant disponible en écriture.

```
private void ParametreOut(out int parametre)
{
    parametre = 5;
}
```

override

Dans une classe dérivée, indique qu'une méthode est redéfinie.

```
public override string ToString()
{
    return "Coucou !";
}
```

params

Indique qu'un paramètre de méthode peut contenir un nombre variable d'éléments.

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

partial

Définit qu'une classe ou une méthode peut avoir plusieurs parties.

```
public partial class ClassePartielle
{
}
```

private

Modificateur de visibilité permettant de rendre un élément visible uniquement par celui qui le contient.

```
private string chaine = "valeur";
```

protected

Modificateur de visibilité permettant de rendre un élément visible uniquement par la classe qui le contient ainsi que ses classes enfants.

```
protected string chaine = "valeur";
```

public

Modificateur de visibilité permettant de rendre un élément visible par tous les types.

```
public string chaine = "valeur";
```

ref

Définit un paramètre de méthode comme étant disponible en lecture et écriture.

```
private void ParametreRef(ref int parametre)
{
    parametre = 42;
}
```

return

Dans une procédure, force le retour à la méthode appelante. Dans une fonction, force le retour à la méthode appelante en renvoyant une valeur.

```
public void Procedure()
{
    return;
}
public int Fonction()
{
```

```
    return 42;
}
```

sbyte

Alias C# du type `System.SByte`. Une variable de ce type contient un numérique entier compris entre -128 et 127.

```
sbyte nombreOiseaux = 123;
```

sealed

Indique qu'une classe ne peut pas être utilisée comme classe de base dans une relation d'héritage.

```
public sealed class ClasseScellee
{
}
```

select

Indique quel résultat doit être retourné par une requête LINQ.

```
var requete = from c in ListeClients
              select c;
```

set

Définit un bloc de code exécuté à l'écriture d'une propriété. Si ce bloc n'est pas défini, la propriété est en lecture seule.

```
public string Message
{
     set { this.message = value; }
}
```

short

Alias C# du type `System.Int16`. Une variable de ce type contient un numérique entier compris entre -32768 et 32767.

```
short nombreOiseaux = 123;
```

static

Indique qu'un membre de classe est lié au type plutôt qu'à une instance du type. Les membres statiques sont utilisés avec la syntaxe `NomDeClasse.Membre`.

L'utilisation du mot-clé `static` avec une classe indique qu'une classe n'est pas instanciable et possède uniquement des membres statiques.

```
public static class ClasseStatique
{
    public static void MethodeStatique
    {
    }
}
```

string

Alias C# du type `System.String`. Une variable de ce type contient une chaîne de caractères.

```
string nomClient = "MARTIN";
```

struct

Définit une structure.

```
public struct MaStructure
{
}
```

switch

Exécute une portion de code en fonction d'une valeur numérique ou d'énumération. Chacun des cas traités est défini par un bloc case.

```
switch (valeur)
{
    case 0:
          Console.WriteLine("Cas 0");
          break;
    case 1:
          Console.WriteLine("Cas 1");
          break;
    default:
          Console.WriteLine("Ni le cas 0, ni le cas 1");
          break;

}
```

this

Référence l'instance actuelle de la classe contenant le code.

```
var objetActuel = this;
```

throw

Permet de générer une exception.

```
throw new Exception("Ceci est une exception !");
```

Utilisé seul dans un bloc `catch`, relance l'exception en cours de traitement.

```
catch (Exception ex)
{
    throw;
}
```

true

Une des deux valeurs pour le type `System.Boolean`.

```
bool estActif = true;
```

try

Définit un bloc d'instructions pouvant générer une exception. Un bloc `try` doit obligatoirement être suivi d'un bloc `catch` et/ou `finally`.

uint

Alias C# du type `System.UInt64`. Une variable de ce type contient un numérique entier compris entre 0 et 18 446 744 073 709 551 615.

```
uint nombreOiseaux = 123;
```

ulong

Alias C# du type `System.UInt64`. Une variable de ce type contient un numérique entier compris entre 0 et 18 446 7440 73 709 551 615.

```
int nombreOiseaux = 123;
```

ushort

Alias C# du type `System.UInt16`. Une variable de ce type contient un numérique entier compris entre 0 et 65 535.

```
int nombreOiseaux = 123;
```

using

En en-tête de fichier, importe un espace de noms dans le fichier courant.

```
using System.Data.SqlClient;
```

Utilisé dans le code, permet de définir un bloc nettoyant automatiquement les ressources d'un objet implémentant l'interface `IDisposable` à la fin de son utilisation.

```
using (SqlConnection connexion = new
SqlConnection("Server=localhost\\SQLEXPRESS;Initial
Catalog=Northwind;Integrated Security=True"))
{
}
```

value

Utilisable uniquement dans le bloc `set` d'une propriété. Il correspond à la valeur assignée à la propriété.

```
public string Message
{
     get { return this.message; }
     set { this.message = value; }
}
```

var

Indique au compilateur qu'il doit déterminer par lui-même le type de la variable. Attention, ne pas confondre avec `dynamic` !

```
var chaine = "Coucou !";
```

virtual

Marque une méthode, une propriété ou un événement comme pouvant être redéfini dans une classe dérivée de la classe courante.

```
public virtual void MethodeVirtuelle()
{
}
```

void

Utilisé comme type de retour pour une procédure.

```
public void MaProcedure()
{
}
```

where

Dans une requête LINQ, permet de filtrer un jeu de données.

```
var requete = from c in ListeClients
              where c.Pays equals "France"
              select c;
```

Permet également de définir une contrainte sur un élément générique.

```
public class Liste<T> where T : IComparable
{
}
```

while

Définit une boucle dont le nombre d'itérations dépend de la valeur retournée par l'évaluation d'une condition.

```
while (i < 50)
{
}
```