**Commentaires : 3 formats**

**//**  ma remarque

/\*  étirage du commentaire sur plusieurs lignes ... \*/

**balises XML**

/// **<summary>**  
 /// *description de la méthode*  
 /// **</summary>**  
 /// **<param name="*param1*">***remarque sur le parametre 1***</param>**

/// **<returns>**remarque sur le retour d'une fonction**</returns>**

**Types de données élémentaires**

|  |  |  |
| --- | --- | --- |
| [*Type*](http://www.gladir.com/CODER/CSHARP/string.htm) | *Octets* | *Domaine* |
| [**byte**](http://www.gladir.com/CODER/CSHARP/byte.htm) | 1 | Nombre naturel de 0 à 255 |
| [**int**](http://www.gladir.com/CODER/CSHARP/int.htm) | 4 | Nombre entier de -2 147 483 648 à 2 147 483 647 |
| [**float**](http://www.gladir.com/CODER/CSHARP/float.htm) | 4 | Nombre réel de 3,4 x 10-38 à 3,4 x 1038 |
| [**long**](http://www.gladir.com/CODER/CSHARP/long.htm) | 8 | Nombre naturel sur 64 bits |
| [**double**](http://www.gladir.com/CODER/CSHARP/double.htm) | 8 | Nombre réel de 1,7 x 10-308 à 1,7 x 10308 |
| [**decimal**](http://www.gladir.com/CODER/CSHARP/decimal.htm) | 16 | Nombre avec un précision de 28 décimals après la virgule |
| **bool** | 1 bit | Booléen de valeur «*true*» ou «*false*». |
| **object** |  | Classe d'objet |
| [**char**](http://www.gladir.com/CODER/CSHARP/char.htm) | 2 | Caractère de type *Unicode*. |
| [**string**](http://www.gladir.com/CODER/CSHARP/string.htm) |  | Chaine de caractères |

**Déclaration d'une constante**

**const** type\_constante nom\_constante = valeur ; //définit constante nom=valeur

**Opérateurs mathématiques Opérateurs relationnels Opérateurs logiques**

|  |  |
| --- | --- |
| **!** | NEGATION binaire |
| **&** | ET binaire |
| **|** | OU binaire |
| **^** | OU EXCLUSIF binaire |
| **&&** | ET logique |
| **||** | OU logique |

|  |  |
| --- | --- |
| [**==**](http://www.gladir.com/CODER/CSHARP/eqeq.htm) | égalité |
| [**!=**](http://www.gladir.com/CODER/CSHARP/noteq.htm) | différence |
| [**>**](http://www.gladir.com/CODER/CSHARP/gt.htm) | plus grand |
| [**>=**](http://www.gladir.com/CODER/CSHARP/gteq.htm) | plus grand ou égal |
| [**<**](http://www.gladir.com/CODER/CSHARP/lt.htm) | plus petit |
| [**<=**](http://www.gladir.com/CODER/CSHARP/lteq.htm) | plus petit ou égal |

|  |  |
| --- | --- |
| [**+**](http://www.gladir.com/CODER/CSHARP/addoperator.htm) | addition |
| [**-**](http://www.gladir.com/CODER/CSHARP/suboperator.htm) | soustraction |
| [**\***](http://www.gladir.com/CODER/CSHARP/muloperator.htm) | multiplication |
| [**/**](http://www.gladir.com/CODER/CSHARP/divoperator.htm) | division |
| [**%**](http://www.gladir.com/CODER/CSHARP/modoperator.htm) | reste d'une division (modulo) |
| **++** | incrémentation |
| **--** | décrémentation |

**Opérateurs sur les chaines de caractères**

|  |  |
| --- | --- |
| **chaine1 + chaine2** | Effectue la concaténation de chaine de caractères |
| **chaine1 == chaine2** | Comparaison d'un égalité |
| **chaine1 != chaine2** | Comparaison d'une différence |
| **chaine1 > chaine2** | Comparaison de plus grand que |
| **chaine1 >= chaine2** | Comparaison de plus grand ou égal que |
| **chaine1 < chaine2** | Comparaison de plus petit que |
| **chaine1 <= chaine2** | Comparaison de plus petit ou égal que |
| **chaine1 += chaine2** | Affectation d'une concaténation de la chaine d'origine |

**Conditionnelles**

**si ... alors ... sinon**

**if** (expression\_vrai)  instruction\_exécutée\_si\_vrai ;

**if** (expression\_vrai)  instruction\_exécutée\_si\_vrai ;

**else**  instruction\_exécutée\_si\_faux ;

**if** (expression\_vrai1)  instruction\_exécutée\_si\_vrai1 ;  
**elseif** (expression\_vrai2)  instruction\_exécutée\_si\_vrai2 ;  
**elseif** (expression\_vrai3)  instruction\_exécutée\_si\_vrai3 ;

**else** instruction\_exécutée\_si\_faux ;

**if** (expression\_vraie1)

**{**

instruction1a\_exécutée\_si\_vrai1 ;

instruction1b\_exécutée\_si\_vrai1 ;

instruction1n\_exécutée\_si\_vrai1 ;

**}**

**elseif** (expression\_vrai2)

**{**

instruction2a\_exécutée\_si\_vrai2 ;

instruction2b\_exécutée\_si\_vrai2 ;

instruction2n\_exécutée\_si\_vrai2 ;

**}**

**elseif** (expression\_vrai3)

**{**

instruction3a\_exécutée\_si\_vrai3 ;

**}**

**else**instruction\_exécutée\_si\_faux ;

**cas selon**

**switch** (variable)

**{**

**case**

*valeur1*: *instruction1\_exécutée\_si\_vrai1*; **break**;

**case**

*valeur2*: *instruction2\_exécutée\_si\_vrai2*; **break**;

**case**

*valeur3*: *instruction1\_exécutée\_si\_vrai3*; **break**;

**default:**

*instruction\_exécutée\_si\_faux*; **break**;

**}**

**switch** (variable)

**{**

**case**  *valeur1*: **case**  *valeur2*:

*instruction12\_exécutée\_si\_vrai1\_ou\_vrai2*; **break**;

**case**  *valeur3*: **case**  *valeur4*: **case**  *valeurn*:

*instruction34n\_exécutée\_si\_vrai3\_ou4\_oun*; **break**;

**default:**

*instruction\_exécuté\_si\_faux*; **break**;

**}**

**Répétitives**

**tant que**

**while** (expression\_vrai)  instruction\_exécutée\_si\_vrai ;

**do**  
**{**

*L'exécution des instructions de la boucle while a lieu* ***au moins une fois***

  instruction1 ;

instruction2 ;

**}**  
**while** (expression\_vrai) ;

**pour**

**for (int i = 1 ; i <=9 ; i++) Console.WriteLine(i);**

*Affichera la suite de 1 ... à 9*

**for** (expression1; expression2; expression3) instruction ;

**for** (expression1; expression2; expression3)

**for (int** i **= 1 ; *i* <=9 ;** i**++)**

**{**

**Console.WriteLine(**i**);**

**Console.WriteLine("On continue la boucle");**

**}**

**{** *// obligatoire si plus d'une instruction à exécuter*

instruction1 ;

instructionp ;

**}**

**pour chaque**

string[] amis=new string[] {"paul","jean","anne","nath"};

**foreach(string nom in amis)**

**{**

**Console.Out.WriteLine(nom);**

**}**

**foreach** (type\_variable variable in liste\_de\_variables)

**{**

instruction1 ;

instructionp ;

**}**

**Tableaux**

**Tableau à une dimension**

**int[] tabEntiers = new int[] {10, 20, 30, 40} ;**

**Console.WriteLine(tabEntiers[0]);**

**Console.WriteLine(tabEntiers[3]);**

**type\_var[]** tableau\_de\_var **= new type\_var[] {**val1, val2, val3, valn**} ;**

**string[] tabMois = new string[12,2] ;**

**tabMois[0,0] = "01";**

**tabMois[0,1] = "Janvier";**

**tabMois[1,1] = "Février";**

**......**

**Console.WriteLine(tabMois[11,11]);**

**type\_var[]** tableau\_de\_var **= new type\_var[**dimension**] ;**

tableau\_de\_var [0] = val1 ;

tableau\_de\_var [1] = val2 ;

....

tableau\_de\_var [3] = valn ;

**Tableau à double dimension**

**string[] tabJours = new string[7] ;**

**tabJours[0] = "Lundi";**

**tabJours[1] = "Mardi";**

**......**

**int nbJours = tabJours.GetLength;**

**Console.WriteLine(nbJours);**

**type\_var[,]** tableau\_de\_variables **= new type\_var[**Lignes,Colonnes**] ;**

tableau\_de\_var[0,0] = val1 ; // ligne 1, colonne 1

tableau\_de\_var[1,1] = val2 ; // ligne 2, colonne 2

....

tableau\_de\_var[n-1, p-1] = valp ; // dernière ligne, dernière colonne

**Remarque**

Les méthodes ***.Length*** et ***.GetLength*** appliquées à un tableau,

permettent de connaître et de récupérer sa dimension

**Gestion des erreurs**

**public static void Main()**  *// début du programme principal*

{

**try**

{

code qui génère l' exception

}

**catch ( IndexOutOfRangeException e1)**

{

traiter l'exception e1

}

**catch ( FormatException e2)**

{

traiter l'exception e2

**}**

**instruction suivante**

*// Programme principal ......................*

**try**

{

//  *… appel de la fonction susceptible de générer une erreur*

}

**catch** **(Exception** *typeException* **e)**

{

*// gestion de l'erreur exécutée en cas d'anomalie programme*

}

**finally** *// la suite sera, dans ce cas, toujours exécutée avec ou sans erreur*

{

*// suite du programme principal ...........................*

} }

**Remarque**

**e** est un objet dérivé du type *Exception*.

Il existe plusieurs types d'exceptions. On peut préciser un ou plusieurs types d'exception tels que *IOException*, *SystemException*,

En écrivant catch (Exception e), on indique qu'on veut gérer toutes les types d'exceptions.

**Les fonctions ... *aujourd'hui les méthodes en POO***

**private static double Hypotenuse(double a, double b)**

**{**

**double sommeDesCarres = a \* a + b \* b;**

**double resultat = Math.Sqrt(sommeDesCarres);**

**return resultat;**

**}**

**public static void Main(string[] args)**

**{**

**double valeur = Hypotenuse(1, 3);**

**Console.WriteLine(valeur);**

**}**

**Retour d'une méthode**

**public static void Main(string[] args)**

{

***type*** Param = ***nomMéthode***(var);

*traitement* Param;

}

**private static *type* *nomMéthode***(**type** **ParamEntrée**)

{

*traitement avec* le *ParamEntrée*

Retour **ParamSortie**

}

**Passage par valeur : Indépendance paramètre effectif et paramètre formel = recopie**

**static void Main(string[] args)**

**{**

**string prenom = "Nicolas";**

**string section = "";**

**DireBonjour(prenom, section);**

**}**

**private static void DireBonjour(string prenom, string section)**

**{**

**prenom = "Maryse";**

**section = "SIO2";**

**Console.WriteLine("Bonjour" + prenom + " en " + section);**

**}**

**public static void Main()**

{

**type** ParamEffectif;

***nomMéthode***(**ParamEffectif**)

}

**private static void *nomMéthode***(ref **type** **ParFormel**)

{

*traitement avec* le *ParFormel*

}

**static void Main(string[] args)**

**{**

**string prenom = "Nicolas";**

**string section = "";**

**ReDireBonjour(ref prenom, ref section);**

**}**

**private static void ReDireBonjour(ref string wprenom, ref string wsection)**

**{**

**wsection = "SIO2";**

**Console.WriteLine("Bonjour" + wprenom + " en " +wsection);**

**}**

**Passage par référence :**

**paramètres effectif et formel sont**

**une seule et même variable en mémoire centrale**