

SL228_POBJ Programmation Objet**Langage C#****Chapitre 5****Méthodes à disposition**

CONTENU DU CHAPITRE 5

5. Les méthodes à disposition	5-1
5.1. Objectifs	5-1
5.2. Syntaxe des méthodes	5-1
5.2.1. Accessibilité	5-1
5.2.2. Méthodes standard dans la classe	5-1
5.2.2.1. Méthodes standards, exemple 1	5-2
5.2.2.2. Utilisation de la méthode static Pow	5-2
5.3. Utilisation des méthodes d'une autre classe	5-3
5.3.1. Utilisation méthodes standard d'une autre classe	5-3
5.3.2. Utilisation Méthodes statique d'une autre classe	5-4
5.4. Paramètre par référence et en sortie	5-5
5.4.1. Le mot clé out	5-5
5.4.1.1. Utilisation out, exemple	5-5
5.4.2. Le mot clé ref	5-6
5.4.2.1. Utilisation ref, exemple	5-6
5.5. Namespace et using	5-7
5.5.1. Using	5-7
5.5.2. Namespace	5-7
5.6. L'espace de noms System	5-8
5.6.1. Les classes des types	5-8
5.6.1.1. Les méthode des classes type	5-8
5.6.1.2. La méthode Parse	5-9
5.6.1.3. La méthode TryParse	5-9
5.6.1.4. La méthode ToString	5-10
5.6.2. System.Math	5-14
5.6.2.1. Les méthodes de System.Math	5-14
5.6.2.2. Les Champs de System.Math	5-16
5.6.3. System.Convert	5-17
5.7. Conclusion	5-18
5.8. Historique des versions	5-18
5.8.1. Version 1.0 Avril 2016	5-18

5. LES METHODES A DISPOSITION

5.1. OBJECTIFS

L'objectif de ce chapitre est de permettre à l'étudiant de réaliser ses propres méthodes dans une classe et surtout d'utiliser les méthodes mise à disposition par le .NET Framework.

5.2. SYNTAXE DES METHODES

Nous allons nous intéresser à la syntaxe de déclaration et réalisation des méthodes standard, et **static**, afin d'avoir aussi un élément de comparaison avec le C++. Nous étudierons aussi le passage de paramètres par référence en C#.

5.2.1. ACCESSIBILITE

Pour déterminer l'accessibilité ou visibilité d'une méthode, elle doit être précédée des mots clés **public**, **protected** ou **private**.

Si la méthode est écrite sans spécifier sa visibilité, par défaut elle est **private**.

5.2.2. METHODES STANDARD DANS LA CLASSE

Prenons comme 1^{er} exemple une méthode événementielle générée automatiquement pour un Button.

```
namespace SyntaxeMethodes
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnTest1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

La méthode btnTest1_Click ne retourne rien d'où **void** et a 2 paramètres.

Nous allons créer une méthode **calcSurfaceCercle** qui calcul la surface d'un cercle en cm² en fonction du rayon fourni en cm.

Voici la nouvelle situation de la classe avec l'utilisation de la méthode calcSurfaceCercle dans la méthode btnTest1_Click. Utilisation de 2 TextBox pour fournir un rayon et afficher le résultat.

5.2.2.1. METHODES STANDARDS, EXEMPLE 1

Voici le nouveau contenu :

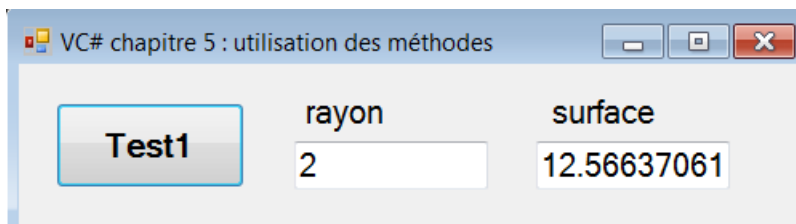
```
namespace SyntaxeMethodes
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnTest1_Click(object sender, EventArgs e)
        {
            double tmpR = double.Parse(txtRayon.Text);
            double tmpS = calcSurfaceCercle(tmpR);
            txtSurface.Text = tmpS.ToString();
        }

        double calcSurfaceCercle (double rayon)
        {
            double surface = Math.Pow(rayon, 2) * Math.PI;
            return (surface);
        }
    }
}
```

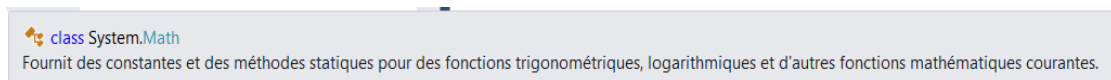
Comme on peut le constater la déclaration et l'utilisation sont très proche du C/C++. Comme l'implémentation n'est pas séparée le nom de la class n'apparaît pas.

Voici un exemple de résultat.

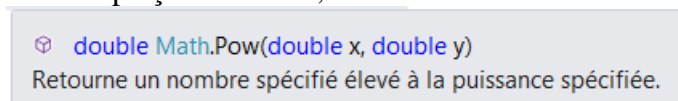


5.2.2.2. UTILISATION DE LA METHODE STATIC POW

Si on s'intéresse à l'expression `Math.Pow` en plaçant le curseur sur `Math`, on obtient :



Et en le plaçant sur `Pow`, on obtient :



Ce qui nous donne le mode d'emploi de la méthode statique `Pow`.

En utilisant l'aide on obtient la déclaration de la méthode Pow:

```
public static double Pow(  
    double x,  
    double y  
)
```

On trouve le mot clé **static** dans la déclaration.

5.3. UTILISATION DES METHODES D'UNE AUTRE CLASSE

Pour illustrer la réalisation et l'utilisation des méthodes standard et statique, il est nécessaire de réaliser une classe définissant ces méthodes et de les utiliser dans une autre.

☹ le mot clé **friend** n'existe pas en C# et n'a pas vraiment d'équivalent !.

Voici le contenu de la classe notre nouvelle classe UtilSurf :

```
namespace SyntaxeMethodes  
{  
    class UtilSurf  
    {  
        public double calcSurfaceCercle(double rayon)  
        {  
            double surface = Math.Pow(rayon, 2) * Math.PI;  
            return (surface);  
        }  
  
        // Méthode statique  
        public static double calcSurfaceCercleS(double rayon)  
        {  
            double surface = Math.Pow(rayon, 2) * Math.PI;  
            return (surface);  
        }  
    }  
}
```

👉 Nécessaire de rendre les méthodes publiques !

5.3.1. UTILISATION METHODES STANDARD D'UNE AUTRE CLASSE

Pour utiliser les méthodes d'une autre classe, il est nécessaire de disposer d'un objet de cette classe. D'où :

```
public partial class Form1 : Form  
{  
    UtilSurf MyUtilSurf = new UtilSurf();  
  
    public Form1()
```

Au niveau de la méthode btnTest1_click, on a :

```
private void btnTest1_Click(object sender, EventArgs e)
{
    double tmpR = double.Parse(txtRayon.Text);
    double tmpS = MyUtilSurf.calcSurfaceCercle(tmpR);
    txtSurface.Text = tmpS.ToString();
}
```

5.3.2. UTILISATION METHODES STATIQUE D'UNE AUTRE CLASSE

Un objet n'est pas nécessaire, au niveau de la méthode btnTest1_click, on a :

```
private void btnTest1_Click(object sender, EventArgs e)
{
    double tmpR = double.Parse(txtRayon.Text);
    double tmpS = UtilSurf.calcSurfaceCercleS(tmpR);
    txtSurface.Text = tmpS.ToString();
}
```

👉 en C# on utilise **UtilSurf.** en C++ on utilise **UtilSurf::.**

Rappel : une méthode statique ne peut pas accéder aux attributs. Si nécessaire il faut lui fournir un objet.

5.4. PARAMETRE PAR REFERENCE ET EN SORTIE

Dans certains cas il est nécessaire de pouvoir modifier une des valeurs passée en paramètre à une méthode.

Pour cela le C# introduits les mots clés **out** et **ref**.

Ils ne sont pas utilisés dans le même contexte et de la même manière.

5.4.1. LE MOT CLE OUT

Il peut arriver qu'un paramètre serve uniquement à récupérer une valeur initialisée par la méthode. C'est notamment utile lorsqu'une méthode doit renvoyer plusieurs valeurs.

Pour ces cas-là, on utilise le mot-clef **out**.

L'utilisation de **out** impose à la méthode d'assigner une valeur au paramètre avant de se terminer. Et contrairement à **ref**, il n'est pas obligatoire d'assigner une valeur au paramètre.

On utilise le mot clé **out** pour permettre à une méthode de fournir plusieurs informations. Par exemple si nous réalisons une méthode `InfosDivInt` qui doit fournir le résultat et le reste d'une division entre deux valeur A et B.

5.4.1.1. UTILISATION OUT, EXEMPLE

Voici dans le cadre de notre projet l'ajout de la méthode `InfoDivInt` qui doit fournir le résultat et le reste d'une division entière entre deux valeur A et B. Ajout pour le test d'un boutons et de 4 TextBox.

```
private int InfoDivInt(int ValA, int ValB, out int Reste)
{
    int Resultat = ValA / ValB;
    // Calcul du Reste
    Reste = ValA % ValB;
    return Resultat;
}
```

La variable `out int Reste` s'utilise normalement dans la méthode. Le compilateur vérifie qu'on affecte bien une valeur.

Utilisation de la méthode dans l'événement du bouton `btnTest2`.

```
private void btnTest2_Click(object sender, EventArgs e)
{
    int tmpResu, tmpReste;
    int tmpA = int.Parse(txtA.Text);
    int tmpB = int.Parse(txtB.Text);

    tmpResu = InfoDivInt(tmpA, tmpB, out tmpReste);

    txtResultat.Text = tmpResu.ToString();
    txtReste.Text = tmpReste.ToString();
}
```

👉 la variable pour obtenir le reste n'a pas besoin d'être initialisée !

Exemple de résultat :

5.4.2. LE MOT CLE REF

Le mot clé **ref** devant le type d'un paramètre indique qu'une variable initialisée doit être fournie lors de l'appel de la fonction, et il indique que cette valeur peut être modifiée par le code exécuté par la fonction.

Attention : ref convient aux variables et aux objets mais :

☛ Les propriétés ne sont pas des variables. Ce sont des méthodes et ne peuvent pas être passées aux paramètres ref.

5.4.2.1. UTILISATION REF, EXEMPLE

Voici dans le cadre de notre projet l'ajout de la méthode AddTva qui permet d'ajouter un taux de tva à un montant. Ajout d'un autre bouton et de 3 TextBox.

```
private void AddTva(double taux, ref double montant)
{
    double valTva = montant * taux;
    montant = montant + valTva;
}
```

La méthode ne retourne rien, elle modifie le montant reçu.

Utilisation de la méthode dans l'événement du bouton btnTest3.

```
private void btnTest3_Click(object sender, EventArgs e)
{
    double tmpTaux = double.Parse(txtTaux.Text) / 100;
    double tmpMontant = double.Parse(txtMHT.Text);
    AddTva(tmpTaux, ref tmpMontant);
    txtMTTC.Text = tmpMontant.ToString();
}
```

Exemple de résultat :

5.5. NAMESPACE ET USING

Si on prend un classique "HelloWorld", mais avec un Button pour déclencher l'action.

```
using System;
using System.Windows.Forms;

namespace HelloWorld
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnTest_Click(object sender, EventArgs e)
        {
            Console.WriteLine("**** Hello world ****");
        }
    }
}
```

5.5.1. USING

La directive Using correspond en quelque sorte au #include du C/C++ et aux import du java.

On remarque que les éléments utilisés par Using sont des namespaces.

```
using System;
using System.Windows.Forms;

namespace HelloWorld
{
    public partial class Form1 : Form
    {
        // namespace System.Windows.Forms
```

5.5.2. NAMESPACE

Le namespace que l'on peut traduire par espace de nom, est un moyen de regrouper des éléments.

On remarque que notre projet HelloWorld devient un namespace.

Il sera possible de l'utiliser dans une solution combinant plusieurs éléments.

5.6. L'ESPACE DE NOMS SYSTEM

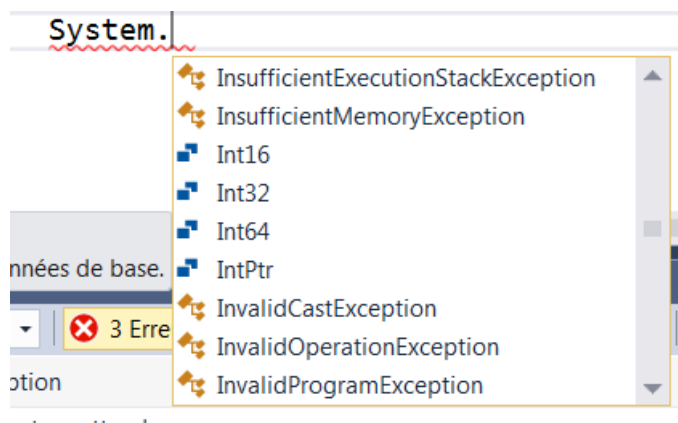
De l'aide on obtient :

L'espace de noms System contient des classes fondamentales et des classes de base qui définissent des types de données valeur et référence, des événements et des gestionnaires d'événements, des interfaces, des attributs, ainsi que des exceptions de traitement couramment utilisés.

D'autres classes fournissent des services prenant en charge la conversion des types de données, la manipulation des paramètres de méthodes, les opérations mathématiques, l'appel de programmes distants et locaux, la gestion de l'environnement d'application, ainsi que le contrôle des applications managées et non managées.

5.6.1. LES CLASSES DES TYPES

Lorsque l'on tape System. une liste déroulante assez longue apparaît :



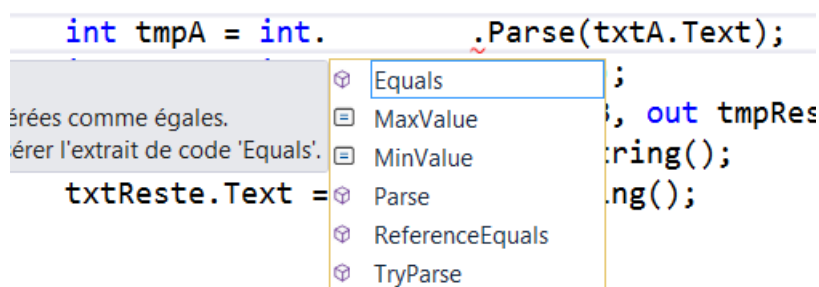
On y trouve par exemple les classes qui correspondent aux types entier.

Rappel les type du C# sont des alias qui représentent les type du .Net Framework

```
int Resultat = ValA / ValB;
// struct System.Int32
// Représente un entier signé 32 bits.
return resultat;
```

5.6.1.1. LES METHODE DES CLASSES TYPE





Lorsque l'on tape int. une liste déroulante apparaît :



On trouve 3 méthodes principales : Parse, TryParse et ToString.

5.6.1.2. LA METHODE PARSE

Elle s'applique à tous les type de base, c'est une méthode surchargée.

	<code>Parse(String)</code>	Convertit la représentation sous forme de chaîne d'un nombre en son équivalent entier 32 bits signé.
	<code>Parse(String, NumberStyles)</code>	Convertit une représentation d'un nombre sous forme de chaîne dans un style spécifié en entier 32 bits signé équivalent.
	<code>Parse(String, IFormatProvider)</code>	Convertit la représentation d'un nombre sous forme de chaîne dans un format propre à une culture spécifié en entier 32 bits signé équivalent.
	<code>Parse(String, NumberStyles, IFormatProvider)</code>	Convertit la représentation sous forme de chaîne d'un nombre dans un style et un format propre à une culture spécifiés en entier 32 bits signé équivalent.

Voici le détail pour les 2 premières surcharges.

5.6.1.2.1. DETAIL PARSE(STRING)

Voici le détail de la méthode Parse(String).

```
public static int Parse(string s)
```

Le **string** `s` doit contenir une expression numérique représentant une valeur entière, si ce n'est pas le cas une exception sera levée.

5.6.1.2.2. DETAIL PARSE(STRING, NUMBERSTYLE)

Voici le détail de la méthode Parse(String, NumberStyle).

```
public static int Parse( string s, NumberStyles style)
```



Le **string** `s` doit contenir une expression numérique représentant une valeur entière avec le style correspondant à une des valeur de **NumberStyles**, si ce n'est pas le cas une exception sera levée.

Par exemple, si on souhaite fournir une expression en hexadécimal on écrira :

```
int inHex = int.Parse(txtInHex.Text,  
System.Globalization.NumberStyles.AllowHexSpecifier);
```

5.6.1.3. LA METHODE TRYPARSE

Elle s'applique à tous les type de base, c'est une méthode surchargée.

	<code>TryParse(String, Int32)</code>	Convertit la représentation sous forme de chaîne d'un nombre en son équivalent entier 32 bits signé. Une valeur de retour indique si la conversion a réussi.
	<code>TryParse(String, NumberStyles, IFormatProvider, Int32)</code>	Convertit la représentation sous forme de chaîne d'un nombre dans un style et un format propre à une culture spécifiés en entier 32 bits signé équivalent. Une valeur de retour indique si la conversion a réussi.

5.6.1.3.1. DETAIL TRYPARSE(STRING, INT32)

Voici le détail de la méthode TryParse(String, Int32).

```
public static bool TryParse(string s, out int result)
```

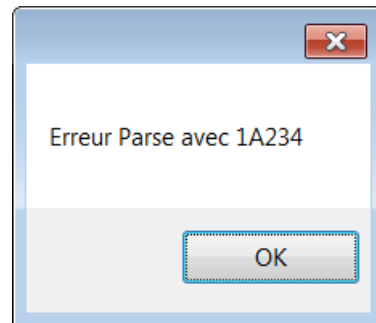
Le **string** s doit contenir une expression numérique représentant une valeur entière, si ce n'est pas le cas la valeur de retour sera false.

La valeur numérique est obtenue avec un paramètre out (référence unidirectionnelle à une variable).

Exemple :

```
if (int.TryParse(txtInHex.Text, out inHex2) == true)
{
    txtLsbHex.Text = inHex2.ToString();
} else {
    txtLsbHex.Text = "Erreur Parse";
    MessageBox.Show("Erreur Parse avec " + txtInHex.Text);
}
```

Si on fournit une valeur qui n'est pas de l'entier décimal on obtient :

**5.6.1.3.1. DETAIL TRYPARSE(STRING, NUMBERSTYLE, IFORMATPROVIDER, INT32)**

Voici le détail de la méthode TryParse(String, NumberStyle, IFormatProvider, Int32).

```
public static bool TryParse(
    string s,
    NumberStyles style,
    IFormatProvider provider,
    out int result
)
```

Le **string** s doit contenir une expression numérique représentant une valeur entière, si ce n'est pas le cas la valeur de retour sera false.

La valeur numérique est obtenue avec un paramètre out (référence unidirectionnelle à une variable).

NumberStyles est une énumération, dont deux éléments sont particulièrement intéressant : **AllowHexSpecifier** et **HexNumber**.

Exemple :

```
if (int.TryParse(txtInHex.Text,
    System.Globalization.NumberStyles.HexNumber,
    System.Globalization.CultureInfo.CurrentCulture,
    out inHex2) == true)
{
    txtLsbHex.Text = inHex2.ToString("X");
}
```

```

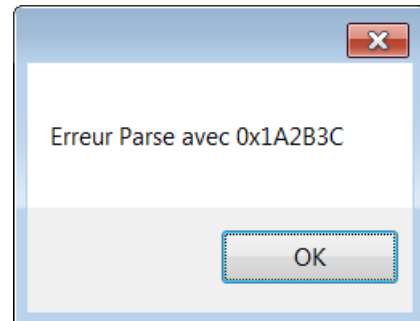
    }
    else
    {
        txtLsbHex.Text = "Erreur Parse";
        MessageBox.Show("Erreur Parse avec " + txtInHex.Text);
    }
}

```

Si on fournit une valeur comme 1A2B3C le résultat est OK.

input Hex	Out lsb Hex
1A2B3C	1A2B3C

Ppar contre avec 0x1A2B3C la conversion n'est pas possible.



5.6.1.4. LA METHODE ToString

Elle s'applique à tous les type de base, c'est une méthode surchargée.

	<code>ToString()</code>	Convertit la valeur numérique de cette instance en sa représentation équivalente sous forme de chaîne. (Substitue <code>ValueType.ToString()</code> .)
	<code>ToString(IFormatProvider)</code>	Convertit la valeur numérique de cette instance en sa représentation sous forme de chaîne équivalente à l'aide des informations de format spécifiques à la culture donnée.
	<code>ToString(String)</code>	Convertit la valeur numérique de cette instance en sa représentation sous forme de chaîne équivalente en utilisant le format spécifié.
	<code>ToString(String, IFormatProvider)</code>	Convertit la valeur numérique de cette instance en sa représentation sous forme de chaîne équivalente à l'aide du format spécifié et des informations de format spécifiques à la culture.

Voici le détail pour la 1^{ère} et la 3^{ème} surcharges.

5.6.1.4.1. DETAIL ToString()

Voici le détail de la méthode ToString().

```
public override string ToString()
```

Appliquée à une valeur numérique entière, cette méthode fournit une chaîne représentant un nombre en décimal.

5.6.1.4.2. DETAIL ToString(String)

Voici le détail de la méthode ToString(String).

```
public string ToString( string format)
```

Pour plus d'informations sur les spécificateurs de format numérique, consultez [Chaînes de format numériques standard](#) et [Chaînes de format numériques personnalisées](#).

Voici le contenu de l'exemple fourni dans l'aide

```
int value = -16325;
string specifier;

// Use standard numeric format specifier.
specifier = "G";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      G: -16325
specifier = "C";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      C: ($16,325.00)
specifier = "D8";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      D8: -00016325
specifier = "E4";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      E4: -1.6325E+004
specifier = "e3";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      e3: -1.633e+004
specifier = "F";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      F: -16325.00
specifier = "N";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      N: -16,325.00
specifier = "P";
Console.WriteLine("{0}: {1}", specifier,
                  (value/100000).ToString(specifier));
// Displays:      P: -16.33 %
specifier = "X";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      X: FFFFC03B

// Use custom numeric format specifiers.
specifier = "0,0.000";
Console.WriteLine("{0}: {1}", specifier, value.ToString(specifier));
// Displays:      0,0.000: -16,325.000
specifier = "#,#.00#; (#,#.00#) ";
Console.WriteLine("{0}: {1}", specifier,
                  (value*-1).ToString(specifier));
// Displays:      #,#.00#; (#,#.00#): 16,325.00
```

On retiendra en particulier le specifier "X" qui fournit la représentation en hexadécimal.

5.6.2. LA METHODE FORMAT DE STRING

La méthode `String.Format`, convertit la valeur des objets en chaînes selon les formats spécifiés et les insère dans une autre chaîne.

La méthode `Format` est surchargée, mais nous nous limitons à la forme :

```
public static string Format( string format, Object arg0)
```

`arg0` étant du type `object`, il est possible de fournir un élément de type différent.

La chaîne de formatage à la syntaxe suivante :

```
{ index[ ,alignment][ :formatString] }
```

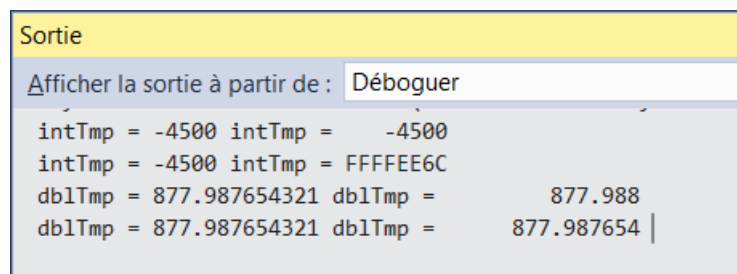
Les possibilités étant très vaste, nous nous limitons à quelques exemples :

5.6.2.1. STRING.FORMAT, EXEMPLES

Voici quelques exemples simples montrant l'utilisation de la méthode `Format` de `String`.

```
string Mess;
int intTmp = -4500;
double dblTmp = 877.987654321;
Mess = "intTmp = " + intTmp +
      String.Format(" intTmp = {0, 8:G}", intTmp);
Console.WriteLine(Mess);
Mess = "intTmp = " + intTmp +
      String.Format(" intTmp = {0, 8:X}", intTmp);
Console.WriteLine(Mess);
Mess = "dblTmp = " + dblTmp +
      String.Format(" dblTmp = {0, 15:F3}", dblTmp);
Console.WriteLine(Mess);
Mess = "dblTmp = " + dblTmp +
      String.Format(" dblTmp = {0, 15:F6}", dblTmp);
Console.WriteLine(Mess);
```

Voici le résultat dans la fenêtre de sortie :



🔗 on trouve un système assez proche des % du bon vieux `printf` du C.

5.6.3. SYSTEM.MATH

Une des sous-section de l'espace de noms System est **Math**, qui contient les fonctions mathématiques et trigonométriques.

5.6.3.1. LES METHODES DE SYSTEM.MATH



Voici la liste des méthodes de Math

Nom	Description
<u>Abs(Double)</u>	Retourne la valeur absolue d'un nombre à virgule flottante double précision.
<u>Abs(Int16)</u>	Retourne la valeur absolue d'un entier signé 16 bits.
<u>Abs(Int32)</u>	Retourne la valeur absolue d'un entier signé 32 bits.
<u>Abs(Int64)</u>	Retourne la valeur absolue d'un entier signé 64 bits.
<u>Abs(SByte)</u>	Retourne la valeur absolue d'un entier signé 8 bits.
<u>Abs(Single)</u>	Retourne la valeur absolue d'un nombre à virgule flottante simple précision.
<u>Acos</u>	Retourne l'angle dont le cosinus est le nombre spécifié.
<u>Asin</u>	Retourne l'angle dont le sinus est le nombre spécifié.
<u>Atan</u>	Retourne l'angle dont la tangente est le nombre spécifié.
<u>Atan2</u>	Retourne l'angle dont la tangente est le quotient de deux nombres spécifiés.
<u>BigMul</u>	Génère le produit intégral de deux nombres 32 bits.
<u>Ceiling(Decimal)</u>	Retourne la plus petite valeur intégrale supérieure ou égale au nombre décimal spécifié.
<u>Ceiling(Double)</u>	Retourne la plus petite valeur intégrale supérieure ou égale au nombre à virgule flottante double précision spécifié.
<u>Cos</u>	Retourne le cosinus de l'angle spécifié.
<u>Cosh</u>	Retourne le cosinus hyperbolique de l'angle spécifié.
<u>DivRem(Int32, Int32, Int32)</u>	Calcule le quotient de deux entiers signés 32 bits et retourne également le reste dans un paramètre de sortie.
<u>DivRem(Int64, Int64, Int64)</u>	Calcule le quotient de deux entiers signés 64 bits et retourne également le reste dans un paramètre de sortie.
<u>Exp</u>	Retourne e élevé à la puissance spécifiée.
<u>Floor(Decimal)</u>	Retourne le plus grand entier inférieur ou égal au nombre décimal spécifié.
<u>Floor(Double)</u>	Retourne le plus grand entier inférieur ou égal au nombre à virgule flottante double précision spécifié.
<u>IEEERemainder</u>	Retourne le reste de la division d'un nombre spécifié par un autre.
<u>Log(Double)</u>	Retourne le logarithme naturel (base e) d'un nombre spécifié.
<u>Log(Double, Double)</u>	Retourne le logarithme d'un nombre spécifié dans une base spécifiée.
<u>Log10</u>	Retourne le logarithme de base 10 d'un nombre spécifié.
<u>Max(Byte, Byte)</u>	Retourne le plus grand de deux entiers non signés 8 bits.
<u>Max(Decimal, Decimal)</u>	Retourne le plus grand de deux nombres décimaux.

<u>Decimal</u>	
<u>Max(Double, Double)</u>	Retourne le plus grand de deux nombres à virgule flottante double précision.
<u>Max(Int16, Int16)</u>	Retourne le plus grand de deux entiers signés 16 bits.
<u>Max(Int32, Int32)</u>	Retourne le plus grand de deux entiers signés 32 bits.
<u>Max(Int64, Int64)</u>	Retourne le plus grand de deux entiers signés 64 bits.
<u>Max(SByte, SByte)</u>	Retourne le plus grand de deux entiers signés 8 bits.
<u>Max(Single, Single)</u>	Retourne le plus grand de deux nombres à virgule flottante simple précision.
<u>Max(UInt16, UInt16)</u>	Retourne le plus grand de deux entiers non signés 16 bits.
<u>Max(UInt32, UInt32)</u>	Retourne le plus grand de deux entiers non signés 32 bits.
<u>Max(UInt64, UInt64)</u>	Retourne le plus grand de deux entiers non signés 64 bits.
<u>Min(Byte, Byte)</u>	Retourne le plus petit de deux entiers non signés 8 bits.
<u>Min(Decimal, Decimal)</u>	Retourne le plus petit de deux nombres décimaux.
<u>Min(Double, Double)</u>	Retourne le plus petit de deux nombres à virgule flottante double précision.
<u>Min(Int16, Int16)</u>	Retourne le plus petit de deux entiers signés 16 bits.
<u>Min(Int32, Int32)</u>	Retourne le plus petit de deux entiers signés 32 bits.
<u>Min(Int64, Int64)</u>	Retourne le plus petit de deux entiers signés 64 bits.
<u>Min(SByte, SByte)</u>	Retourne le plus petit de deux entiers signés 8 bits.
<u>Min(Single, Single)</u>	Retourne le plus petit de deux nombres à virgule flottante simple précision.
<u>Min(UInt16, UInt16)</u>	Retourne le plus petit de deux entiers non signés 16 bits.
<u>Min(UInt32, UInt32)</u>	Retourne le plus petit de deux entiers non signés 32 bits.
<u>Min(UInt64, UInt64)</u>	Retourne le plus petit de deux entiers non signés 64 bits.
<u>Pow</u>	Retourne un nombre spécifié élevé à la puissance spécifiée.
<u>Round(Decimal)</u>	Arrondit une valeur décimale à la valeur entière la plus proche.
<u>Round(Double)</u>	Arrondit une valeur à virgule flottante double précision à la valeur entière la plus proche.
<u>Round(Decimal, Int32)</u>	Arrondit une valeur décimale au nombre de chiffres fractionnaires spécifié.
<u>Round(Decimal, MidpointRounding)</u>	Arrondit une valeur décimale à l'entier le plus proche. Un paramètre spécifie comment arrondir la valeur qui se trouve à égale distance des deux nombres.
<u>Round(Double, Int32)</u>	Arrondit une valeur à virgule flottante double précision au nombre de chiffres fractionnaires spécifié.
<u>Round(Double, MidpointRounding)</u>	Arrondit une valeur à virgule flottante double précision à l'entier le plus proche. Un paramètre spécifie comment arrondir la valeur qui se trouve à égale distance des deux nombres.
<u>Round(Decimal, ...)</u>	Arrondit une valeur décimale au nombre de chiffres fractionnaires

<u>Int32, MidpointRounding</u>	spécifié. Un paramètre spécifie comment arrondir la valeur qui se trouve à égale distance des deux nombres.
<u>Round(Double, Int32, MidpointRounding)</u>	Arrondit une valeur à virgule flottante double précision au nombre de chiffres fractionnaires spécifié. Un paramètre spécifie comment arrondir la valeur qui se trouve à égale distance des deux nombres.
<u>Sign(Decimal)</u>	Retourne une valeur indiquant le signe d'un nombre décimal.
<u>Sign(Double)</u>	Retourne une valeur indiquant le signe d'un nombre à virgule flottante double précision.
<u>Sign(Int16)</u>	Retourne une valeur indiquant le signe d'un entier signé 16 bits.
<u>Sign(Int32)</u>	Retourne une valeur indiquant le signe d'un entier signé 32 bits.
<u>Sign(Int64)</u>	Retourne une valeur indiquant le signe d'un entier signé 64 bits.
<u>Sign(SByte)</u>	Retourne une valeur indiquant le signe d'un entier signé 8 bits.
<u>Sign(Single)</u>	Retourne une valeur indiquant le signe d'un nombre à virgule flottante simple précision.
<u>Sin</u>	Retourne le sinus de l'angle spécifié.
<u>Sinh</u>	Retourne le sinus hyperbolique de l'angle spécifié.
<u>Sqrt</u>	Retourne la racine carrée d'un nombre spécifié.
<u>Tan</u>	Retourne la tangente de l'angle spécifié.
<u>Tanh</u>	Retourne la tangente hyperbolique de l'angle spécifié.
<u>Truncate(Decimal)</u>	Calcule la partie entière d'un nombre décimal spécifié.
<u>Truncate(Double)</u>	Calcule la partie entière d'un nombre à virgule flottante double précision spécifié.

5.6.3.2. LES CHAMPS DE SYSTEM.MATH

	Nom	Description
	E	Représente la base logarithmique naturelle spécifiée par la constante e .
	PI	Représente le rapport de la circonférence d'un cercle à son diamètre, spécifié par la constante π .

5.6.4. SYSTEM.CONVERT

5.7. CONCLUSION

A l'issue de ce chapitre, l'étudiant, aura découvert comment utiliser et créer des méthodes. Il aura aussi une idée de l'ampleur des fonctions à disposition.

5.8. HISTORIQUE DES VERSIONS

5.8.1. VERSION 1.0 AVRIL 2016

Création du document.