Type du C# de base

Type C#	Type .NET Framework	Plage	Taille
bool	System.Boolean	false(0) true(1 …)	8 bits
char	System.Char	U+0000 à U+ffff	Caractère Unicode 16 bits
byte	System.Byte	0 à 255	Entier 8 bits non signé
sbyte	System.SByte	-128 à 127	Entier 8 bits signé
short	System.Int16	-32 768 à 32 767	Entier 16 bits signé
ushort	System.UInt16	0 à 65 535	Entier 16 bits non signé
int	System.Int32	-2 147 483 648 à 2 147 483 647	Entier 32 bits signé
uint	System.UInt32	0 à 4 294 967 295	Entier 32 bits non signé
long	System.Int64	-9,223,372,036,854,775,808 à 9,223,372,036,854,775,807	Entier 64 bits signé
ulong	System.UInt64	0 à 18,446,744,073,709,551,615	Entier 64 bits non signé
float	System.Single	±1,5e−45 à ±3,4e38 7 chiffres	32 bits
double	System.Double	±5,0e−324 à ±1,7e308 15-16 chiffres	64 bits
decimal	System.Decimal	-7.9 x 10e28 à 7.9 x 10e28 28-29 chiffres	128 bits
string	System.String		chaine de caractères

Type complexe

Type personnalisable : package de types de base (ex : date de naissance, etc.). On lui donne un nom personnel et on place n'importe quel types de base à l'intérieur (appelés champs).

```
struct DateDeNaissance
{
    public Byte Jour;
    public Byte Mois;
    public UInt16 Annee;
}
```

On l'utilise de base en déclaration et en définition on utilise le . pour accéder aux champs.

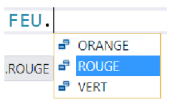
```
DateDeNaissance Joseph;
Joseph.Jour = 30;
Joseph.Mois = 12;
Joseph.Annee = 1985;
Console.WriteLine(Joseph.Jour + "/" + Joseph.Mois + "/" + Joseph.Annee);
```

Type énuméré

Fabrication de l'énumération

```
enum FEU { ROUGE, ORANGE, VERT}
```

Utilisation




```
FEU FeuRueDesDombes;
FeuRueDesDombes = FEU.ROUGE;

Console.WriteLine(FeuRueDesDombes);
```

Les tableaux statiques (taille fixe)

Ils contiennent un ensemble d'élément de même type (y compris structure et enum)

Exemple d'un tableau nommé « tab » de 5 entiers (Int32)

```
Int32[] tab = new Int32[5];
tab[0] = 36;
tab[4] = 8;
```

```
Attention : les index vont toujours de 0 à taille-1
```

```
Int32[] toto = {23, 5, -9}; // tableau prédéfini et tjs modifiable
Int32 [,] tab2 = new Int32 [5,3];     //tableau 2D
tab2[4, 2] = 23;
Int32 [,,] tab3 = new Int32[5, 3, 4]; //tableau 3D
```

Les tableaux dynamiques (taille adaptable)

```
List<Int32> maListe = new List<Int32>();
maListe.Add(39);
maListe.Add(45);
maListe.Add(12);
maListe.Sort();
```

Les opérateurs

- + - * / %(modulo : reste entier d'une division ex : 5%2 →1)
- x=x+1 → x++ (de même existe pour x--)

  Rq : y=x++ : post incrémentation (y = x et x sera incrémenté de 1)
  Y=++x : pré incrémentation (x est incrémenté et y prendra cette valeur incrémentée)
- `b += a ; équivalent à b = b + a ;`
- ! (inversion) fonctionne que pour les Boolean y= !x
- & | ^ et, ou ou exclusif logique

```
Int32 a = 0x1F;
Int32 b = Convert.ToInt32("00100101", 2);
Int32 c = a & b;
```

- `b = a << 1 ; on décale en binaire la valeur de a de 2 bits vers la gauche`

Les conditions et itérations

```
opérateur de test : >    <    >=    <=    == égalité    != différent    &&    ||
```

- si /sinon : if else

```
if (n1 > n2)
    Console.WriteLine(n1 + " est supérieur à " + n2);
else if (n1 < n2)
    Console.WriteLine(n1 + " est inférieur à " + n2);
else
    Console.WriteLine("les 2 nombres sont égaux");
```

- étude de cas : switch case

```
switch (n1)
{
    case 0:
        Console.WriteLine("Tu es nul est le restera");
        break;
    case 20:
        Console.WriteLine("Tu es au TOP");
        break;
    default:
        Console.WriteLine("Tu peux progresser");
        break;
}
```

- boucle définie : for

```
for (i = 0; i < 10; i = i + 2)    // 0 2 4 6 8
{
    Console.WriteLine(i);
}
```

- boucle faire tant que : do while

```
i = 0;
do
{
    i++;
    Console.WriteLine(i);         // 1 2 3 4 5 6 7 8 9 10
} while (i < 10);
```

- boucle tant que, je fais : while

```
i = 0;
while (i < 10)
{
    i++;
    Console.WriteLine(i);         // 1 2 3 4 5 6 7 8 9 10
}
```

- foreach (pour chaque élément d'un tableau)

```
Int32[] toto = {23, 5, -9};
foreach (Int32 x in toto)
    Console.WriteLine(x);
```