Typage avancé en C#

- Signed et Unsigned
- decimal et float
- Long, Short et Tiny
- BigInteger
- DateTime
- Timespan

Signed / Unsigned

Signed (avec signe)

- Les types **signed** peuvent représenter des nombres positifs et négatifs.
- Exemples : `int`, `long`, `short`, `sbyte`.

```
int nombreSigned = -100;  // Représente un entier négatif
```

Unsigned (sans signe)

- Les types **unsigned** ne peuvent représenter que des nombres positifs.
- Exemples : `uint`, `ulong`, `ushort`, `byte`.

```
uint nombreUnsigned = 100;  // Représente seulement un entier positif
```

Signed / Unsigned

Convertion

Les convertions *ToInt32*, *ToDouble*, etc ont tous leur équivalent *Unsigned*.

```
uint a = Convert.ToUInt32(Console.ReadLine());
Console.WriteLine(a);
```

Exception

Si on essaye de forcer une valeur négative à un Uint, nous aurons une exception.

```
uint a = Convert.ToUInt32(Console.ReadLine()); // Si l'utilisateur saisie -3
Console.WriteLine(a); // System.OverflowException: Value was either too large or too small for a UInt32.
```

Limite de taille en fonction du type.

- Lors de la déclaration d'une variable, lorsqu'on choisit un type de donnée, on détermine en faites la quantité d'espace mémoire à réserver stocker la valeur.
    - Cela induit une limite de taille à la valeur qu'on peut y stocker.
- Dans le cas d'un entier (int) on réserve 32 bit de mémoire. 16bit négatif et 16 bit positif.
- Dans le cas d'un entier positif (Uint), on réserve la même quantité de mémoire (32 bit), mais cette fois, entièrement alloué au nombre positif, ce qui permet d'avoir une limite positive plus élevé.
    - int: -2,147,483,648 à 2,147,483,647
    - uint: 0 to 4,294,967,295
- Insérer un valeur plus grande ou plus petite que la limite déclenche l'exception:

```
OverflowException: Value was either too large or too small for a UInt32.
```

Decimal vs float

`decimal`

- Utilisé pour des valeurs monétaires ou précises.
- Plus de précision (28-29 chiffres significatifs) car il réserve 128 bits d'espace (contre 64 bits pour le double)
- Plus lent que `float` et `double`.

```
decimal montant = 12345.67m; // notez l'ajout du m à la fin pour signifier que c'est une donnée de type décimal.
```

Exemples d'utilisation :

- Calculs financiers, comme les taux d'intérêt, les montants d'argent, ou la comptabilité.
- Travailler avec des prix ou des monnaies où chaque centime compte et aucune erreur d'arrondi n'est acceptable.

`float`

- Utilisé pour des calculs scientifiques où une petite précision peut être perdue (ex. afficher sur un graphique)
- 7 chiffres significatifs car il réserve 32 bits d'espace (contre 64 bits pour le double)
- Moins précis que `decimal` mais plus rapide.

```
float temperature = 36.6f; // notez l'ajout du f à la fin pour signifier que c'est une donnée de type float.
```

Exemples d'utilisation :

- Graphiques 3D (comme la modélisation ou les jeux vidéo).
- Simulations physiques, où la précision relative est moins importante.
- Travailler avec des capteurs ou des données en temps réel dans des systèmes embarqués, où l'efficacité en mémoire est essentielle.

Long / Short / byte

`long`

- Représente un entier 64 bits avec signe (de -2^63 à 2^63-1).

```
long grandNombre = 9223372036854775807L;
```

`short`

- Représente un entier 16 bits avec signe (de -32 768 à 32 767).

```
short petitNombre = 32767;
```

`byte` (Tiny)

- Représente un entier 8 bits sans signe (de 0 à 255).

```
byte tinyByte = 255;
```

BigInteger

`BigInteger`

- Utilisé pour représenter des nombres très grands, sans limite pratique.
- Utilise le même principe qu'une string, mais appliqué au nombre, pour permettre d'avoir une taille maximale flexible.
- BigInteger n'est pas un type natif, il vient avec la librairie Numerics
  - *L'utilisation de bibliothèque sera couverte plus tard dans la session.*

```
using System.Numerics; // Importer la librairie Numerics

BigInteger grandNombre = BigInteger.Parse("12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890");
```

- Pas de limite de taille, contrairement à `long`.

Date / Time / DateTime

`DateTime`

- Combine à la fois la **date** et l'**heure**.
- Certain language de programmation ont une classe Date et une classe Time pour gérer ceux-ci séparement.
    - Ce n'est pas le cas en C#, où tout est dans la classe DateTime.

```
DateTime maintenant = DateTime.Now;
Console.WriteLine(maintenant); // 2024-10-17 16:14:46

DateTime birthDate = new DateTime(1990, 5, 25, 12, 5, 0);
Console.WriteLine(birthDate); // 1990-05-25 12:05:00

DateTime specificDate = new DateTime(2023, 10, 17);
Console.WriteLine(specificDate); // 2023-10-17 00:00:00
```

Methods de la classe `DateTime`

La classe DateTime nous offre une panoplie de méthodes pour travailler avec les dates.

```
DateTime today = DateTime.Today;
Console.WriteLine(today); // 2024-10-17 00:00:00

DateTime futureDate = today.AddDays(10); // Existe aussi AddHours, AddMinutes, AddMounth, etc..
Console.WriteLine(futureDate); // 2024-10-27 00:00:00

DayOfWeek day = today.DayOfWeek; // Thursday
Console.WriteLine(day);
```

`DateTime` Chainage d'opération.

- Puisque les méthodes de classe DateTime retourne une nouvelle instance de celle-ci, nous pouvons les chaîner pour faire plusieurs opérations d'un seul coup.

```
DateTime now = DateTime.Now;
Console.WriteLine(now); // 2024-10-17 16:17:17
DateTime futureTime = now.AddHours(3).AddMinutes(30);
Console.WriteLine(futureTime); // 2024-10-17 19:47:17
```

`DateTime`

- Il y a plusieurs autres méthode de disponibles pour manipuler les dates et les heures.
- N'hésitez pas à rechercher dans la documentation en fonction de vos besoins.

[https://learn.microsoft.com/en-us/dotnet/api/system.datetime?view=net-8.0#methods]

Formater les dates et heures C#

Formater une date (`DateTime`)

- Lorsqu'on veut afficher l'information dans la console, on peut la convertir en string et spécifier le formatage à utiliser.
- La méthode `ToString` permet de formater des dates en utilisant des spécificateurs standard ou personnalisés.

Exemple de base :

```
DateTime now = DateTime.Now;
string formattedDate = now.ToString("dd/MM/yyyy");
Console.WriteLine(formattedDate); // 17/10/2024
```

- **d**: 6/15/2008
- **D**: Sunday, June 15, 2008
- **f**: Sunday, June 15, 2008 9:15 PM
- **F**: Sunday, June 15, 2008 9:15:07 PM
- **g**: 6/15/2008 9:15 PM
- **G**: 6/15/2008 9:15:07 PM
- **m**: June 15
- **o**: 2008-06-15T21:15:07.0000000
- **R**: Sun, 15 Jun 2008 21:15:07 GMT
- **s**: 2008-06-15T21:15:07
- **t**: 9:15 PM
- **T**: 9:15:07 PM
- **u**: 2008-06-15 21:15:07Z
- **U**: Monday, June 16, 2008 4:15:07 AM

Format personnalisé pour la date

Pour des formats plus spécifiques, vous pouvez utiliser des chaînes personnalisées.

Exemple :

```
DateTime now = DateTime.Now;
string customFormat = now.ToString("yyyy-MM-dd HH:mm:ss");
Console.WriteLine(customFormat); // 2024-10-17 14:35:24
```

Spécificateurs de format personnalisés :

'h:mm:ss.ff t': 9:15:07.00 P

'd MMM yyyy': 15 Jun 2008

'HH:mm:ss.f': 21:15:07.0

'dd MMM HH:mm:ss': 15 Jun 21:15:07

Formater une combinaison date et heure (`DateTime`)

Vous pouvez formater des dates et heures ensemble avec les mêmes spécificateurs.

Exemple de format long avec date et heure :

```
DateTime now = DateTime.Now;
string formattedDateTime = now.ToString("F");
Console.WriteLine(formattedDateTime); // Thursday, 17 October 2024 14:35:24
```

Exemple avec un format personnalisé :

```
DateTime now = DateTime.Now;
string customFormat = now.ToString("yyyy-MM-dd HH:mm:ss");
Console.WriteLine(customFormat); // 2024-10-17 14:35:24
```

DateTime

- Pour conclure, les possibilités avec les dates, heures et durées sont nombreuses.
- N'hésitez pas à retourner vous référer à la documentation lorsque vous les utiliserez.

TimeSpan

`TimeSpan`

- Représente une durée de temps, pas un point dans le temps. On peut l'utiliser pour les calculs sur des durées.

```
TimeSpan duration = new TimeSpan(2, 14, 18);
Console.WriteLine(duration); // 02:14:18
TimeSpan duree = TimeSpan.FromHours(1.5);
Console.WriteLine(duree); // 01:30:00

TimeSpan additionalTime = duration.Add(new TimeSpan(1, 0, 0));
Console.WriteLine(additionalTime); // 03:14:18

TimeSpan reducedTime = duration.Subtract(new TimeSpan(0, 30, 0));
Console.WriteLine(reducedTime); // 01:44:18

bool isLonger = duration > additionalTime;
Console.WriteLine(isLonger); // False
```

Formater une durée (`TimeSpan`)

La classe `TimeSpan` est utilisée pour représenter des durées et intervalles de temps.

Exemple de base :

```
TimeSpan duration = new TimeSpan(2, 14, 18); // 2h 14m 18s
string formattedDuration = duration.ToString();
Console.WriteLine(formattedDuration); // 02:14:18
```

Format personnalisé pour `TimeSpan` :

- **"c"** : Format court (ex. `hh:mm:ss`)
- **"g"** : Format général court (ex. `2:14:18`)
- **"G"** : Format général long (inclut les jours, ex. `0:02:14:18.1234567`)

Exemple avec un format personnalisé :

```
TimeSpan duration = new TimeSpan(1, 5, 30, 45); // 1 jour, 5h, 30m, 45s
string customDuration = duration.ToString("dd:hh:mm:ss");
Console.WriteLine(customDuration); // 01:05:30:45
```