



Initiation C#

Installation du module application console C# dans visual studio

1. Ouvrir visual studio
2. Essayer de créer un nouveau projet
3. Ouvrir visual studio installer
4. Installer Développement .NET DESKTOP et Développement pour la plateforme Windows Universelle
5. Pour des raisons pratiques, il est conseillé de faire ces installations avant le début du cours.

Introduction



C'est un langage de
Programmation Orientée Objet



C'est un dérivé du C



Crée par Microsoft en 2002



Les instructions



Un programme est composé d'instructions.

Chaque instruction est une opération effectuée par l'ordinateur.

Il y a 3 types d'instructions essentiels : la déclaration, l'assignation, la lecture.

Chaque instruction est suivi d'un « ; ».



Déclaration

- Permet de créer une variable.
- En fonction de l'endroit où la déclaration est effectuée, l'accessibilité de l'objet déclarée change.



Déclaration de variables

```
1 public int index = 0;  
private2 bool3 test =4 false;  
...
```

- 1 : modificateur d'accès
(access modifier)

- 2 : type

- 3 : nom de variable

- 4 : valeur d'initialisation ou
d'assignation

Les variables

- Elles servent à stocker des valeurs
- Une variable ne peut stocker que des valeurs de son type
- Si le résultat d'une opération n'est pas stocké dans une variable il est perdu.
- Les principaux types de variables : int, float, double, string, bool

Les type et variables

- En c# **tout est objet**. Les types primitifs sont des enfants de Object

En c# les types sont séparés en deux grande catégories :

- les types valeur (value type)
- les types référence (reference type)

Types de valeur

- [Types simples](#)
 - [Intégral signé](#): sbyte , short , int , long
 - [Entier non signé](#): byte , ushort , uint , ulong
 - [Caractères Unicode](#) : char , qui représente une unité de code UTF-16
 - [Virgule flottante binaire IEEE](#): float , double
 - [Virgule flottante décimale haute précision](#): decimal
 - Boolean : bool , qui représente des valeurs booléennes, les valeurs qui sont true ou false
- [Types ENUM](#)
 - Types définis par l'utilisateur sous la forme enum E {...} . Un type enum est un type distinct avec des constantes nommées. Chaque type enum a un type sous-jacent qui doit être un des huit types intégraux. L'ensemble de valeurs d'un type enum est le même que l'ensemble de valeurs du type sous-jacent.
- [Types struct](#)
 - Types définis par l'utilisateur de la forme struct S {...}
- [Types valeur Nullable](#)
 - Extensions de tous les autres types de valeurs avec une valeur null
- [Types de valeur de Tuple](#)
 - Types définis par l'utilisateur de la forme (T1, T2, ...)

Assignment

- Permet de modifier la valeur stockée dans une variable.
- `S = 5 ;`



Lecture

- Permet d'accéder à la valeur stockée dans une variable.
- Il suffit d'utiliser le nom de la variable.



%

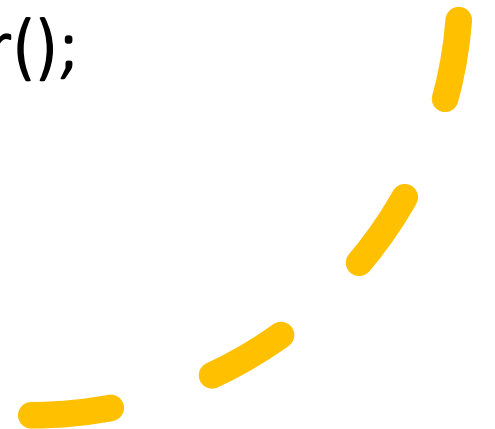
- $A \% B$ renvoie le reste de la division euclidienne de A par B .
- Si $A \% B$ est égal à 0 alors A est un multiple de B .
- Il est souvent utilisé pour créer des comportements cycliques.



```
1      using System;
2
3      namespace Initiation
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  Console.WriteLine("Hello World!");
10             }
11         }
12     }
```

IHM

- Afficher dans la console :
`Console.WriteLine(« HelloWorld »);`
- Lecture des input utilisateur dans la console:
`Console.ReadLine();`
- Effacer la console : `Console.Clear();`





La concaténation

```
string s1 = «monsieur»;  
string s2 = «le président.»;  
Console.WriteLine(« Bonjour {0}  
{1} », s1, s2);
```

Résultat:

Bonjour monsieur le président.



Transformer
une string en
int ou float

```
string s1 = «1»;  
string s2 = «2»;
```

```
int i1 = int.Parse(s1);  
int i2 = int.Parse(s2);  
int resultat = i1 + i2;
```

```
Console.WriteLine(« Le résultat est  
{0}. » ,resultat);
```

Affichage console:
Le résultat est 3.



Cast

- Changer le type d'un objet
- Si l'objet n'est pas convertible dans le type voulu alors Exception

```
static void Main(string[] args)
{
    int a = 5;
    float b = (float)a; // convert a to float
}
```



Exercice 1

- Faire un programme qui demande le nom et le prénom de l'utilisateur puis affiche : « Bonjour prénom nom. »



Exercice 2

- Faire un programme qui demande l'année de naissance d'une personne et qui affiche son âge
- Faire en sorte que le programme demande aussi le jour et le mois. Utiliser cette information pour donner un âge avec plus de justesse.



Exercice 3

- Saisir le prix hors taxe d'un produit puis afficher le prix du produit avec la TVA. On considère que la TVA est de 20%. Par exemple si vous rentrez un prix de 100 euros, le prix avec la TVA est de 120 euros.
- Saisir aussi le taux de TVA en prenant en compte que ce n'est pas forcément un entier. Modifier le programme pour utiliser le taux de TVA saisi.



Les fonctions

Réutiliser plus facilement une série d'instructions

Code plus compact, plus facilement maintenable ou transférable.

Utilisation de fonctions

0 référence

```
public static void Main(string[] args) 1
{
    HelloWorld(); 2
}
```

3

1 référence

```
public static void HelloWorld()
{
    6 Console.WriteLine("Hello World!");
}
```

5 4

- 1 : signature de méthode
- 2 : arguments
- 3 : appel de fonction
- 4 : nom de la fonction
- 5 : type de retour
- 6 : corps de méthode



Exercice 4

- Créez une fonction qui permet de faire une addition entre 2 nombres. Elle doit afficher dans la console : $\text{Nombre1} + \text{Nombre 2} = \text{Résultat}$.
- Créez une fonction qui permet de gérer la division euclidienne entre 2 nombres. Elle doit afficher : $\text{DIVIDENDE} / \text{DIVISEUR} = \text{QUOTIENT}$ reste RESTE .
- Créer un programme qui permet à l'utilisateur de taper 2 nombres puis qui affiche le résultat de l'addition, puis de la division de ces 2 nombres en utilisant les fonction créées précédemment.

Les tests

```
if (condition) { action }
```

```
else if(condition) {action}
```

```
else (condition) { action }
```

Les conditions

$X > 5$

$3 \leq 5$

$Y == 0$

$1 \neq 1$

Les opérateurs logiques

Il est possible de mettre plusieurs conditions dans un if.

Pour pouvoir lier plusieurs conditions, il faut utiliser des opérateurs logiques.

&&

&& (AND) est un opérateur logique qui renvoie vrai si toutes les conditions reliées sont vraies.

Condition1 : $1 < 2$ \Rightarrow vrai

Condition2 : $0 \geq 0$ \Rightarrow vrai

Condition3 : $5 < 4$ \Rightarrow faux

Condition1 && Condition2 \Rightarrow vrai

Condition1 && Condition3 \Rightarrow faux

||

|| (OR) est un opérateur logique qui renvoie vrai si au moins l'une des conditions reliées est vraie.

Condition1 : $3 < 2$ \Rightarrow faux

Condition2 : $0 \neq 5$ \Rightarrow vrai

Condition3 : $8 < 3$ \Rightarrow faux

Condition1 || Condition2 \Rightarrow vrai

Condition1 || Condition3 \Rightarrow faux

!

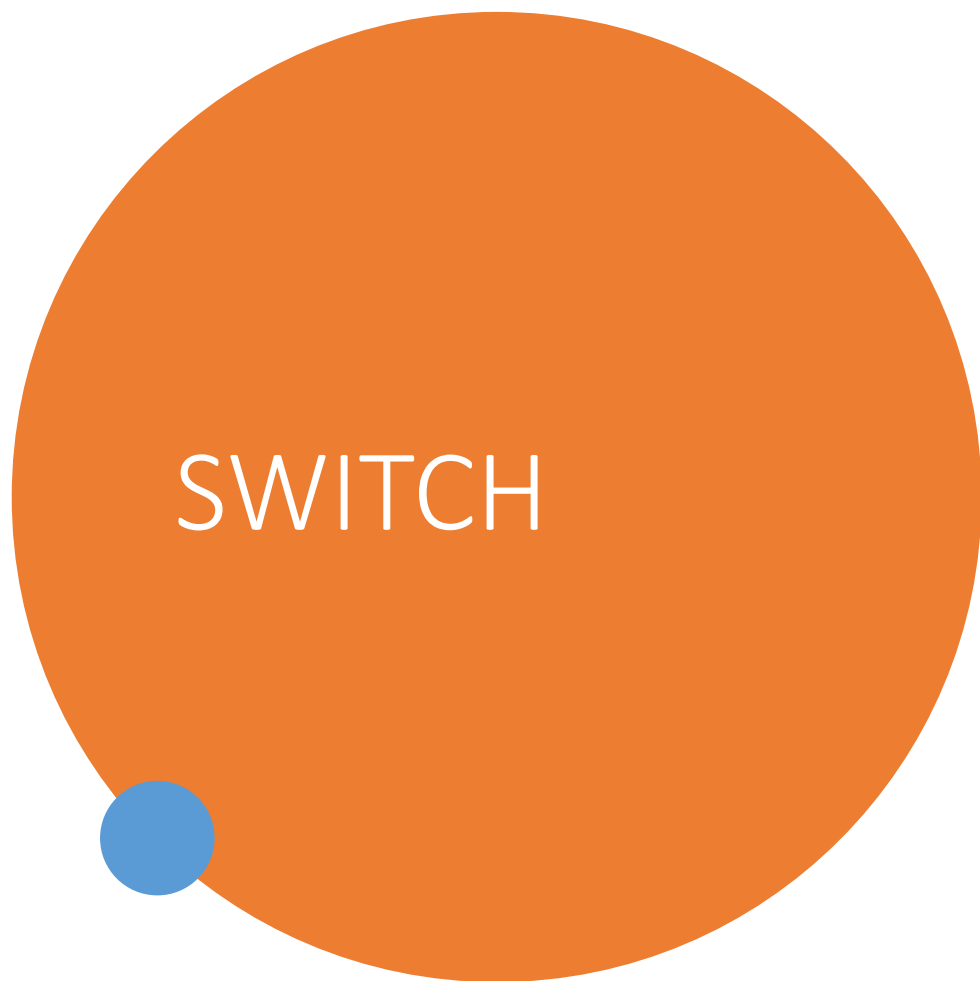
! (NOT) est un opérateur logique qui renvoie le contraire d'un résultat d'une condition.

Condition1 : $3 < 2$ \Rightarrow faux

Condition2 : $0 \neq 5$ \Rightarrow vrai

!Condition1 \Rightarrow vrai

!Condition2 \Rightarrow faux



```
switch(value)
{
    case 1 :
        //instructions
        break;
    case 2 :
        //instructions
        break;
    default :
        //instructions
        break;
}
```






Exercice 5

- Tester la division par 0 dans la fonction de l'exercice 4.
- Tester de ne pas taper un nombre dans les fonctions de l'exercice 4.
- Utiliser des tests pour gérer ces cas et afficher un message d'erreur personnalisé.



Exercice 6

- Ecrire le programme qui change la couleur d'un feu rouge à partir de son état courant {VERT, ORANGE, ROUGE}.
- 



Exercice 7

- Ecrire un programme qui affiche l'état de l'eau en fonction de sa température.

Gestion des exceptions

- Une exception est une manipulation inattendue qui interrompt l'exécution d'un programme.
- Le bloc Try englobe le bout de code susceptible de lever une exception.
- Le bloc Catch permet d'intercepter une exception et de la gérer en laissant le programme continuer à tourner.

```
using System;

namespace CodeSamples
{
    0 références
    class Program
    {
        0 références
        static void Main(string[] args)
        {
            string s = "coucou";
            try
            {
                int number = int.Parse(s);
            }
            catch (Exception e)
            {
                Console.WriteLine("Error parsing s. Message = {0}", e.Message);
            }
        }
    }
}

//result : Error parsing s. Message = Input string was not in a correct format.
```



Exercice 8

- Utiliser try / catch pour gérer les valeurs incorrects dans une nouvelle version de vos fonctions add et divide.



Exercice 9

- rentrer 3 nombres.
- Calculer 3 nombres aléatoires entre 1 et 100.
- afficher ces 6 nombres dans l'ordre décroissant.
- Gérer tous les cas particuliers dans lesquels le nombre n'est pas rentré correctement.



Exercice 10

- Ecrire le programme qui classe par ordre croissant 3 données réelles 'a', 'b', 'c', tapée par l'utilisateur.
- Gérer tous les cas particuliers pour lesquels les inputs ne sont pas corrects.

Les boucles

Les boucles servent à répéter une opération sur un ensemble de données

Une boucle s'exécute tant que sa condition d'exécution est vraie

for (utilisée quand on connaît le nombre de répétitions)

while (utilisée quand on ne connaît pas le nombre de répétitions)

Foreach (utilisée pour parcourir entièrement un tableau ou une liste)

for

```
for( int i = 0; i < 5; i++)  
{  
    //corps de la boucle  
    Console.WriteLine(« itération n»+i);  
}
```





Exercice 11

- Afficher les multiples de 2 entre 0 et 20 (inclus) dans l'ordre croissant avec une boucle for.
- Chaque multiple doit être affiché sur une nouvelle ligne.



Exercice 12

- Saisir un nombre puis afficher la somme totale en additionnant tous les nombres jusqu'à ce nombre.
- Exemple si vous saisissez 7. Vous devez afficher le résultat de $1 + 2 + 3 + 4 + 5 + 6 + 7$. le résultat pour 7 est 28.



Exercice 13

- Ecrire tous les multiples de 3 entre 0 et 30 dans l'ordre décroissant.
- Les multiples doivent être affichés sur la même ligne séparés par des tirets.

while

```
int i = 0;  
While(i < 5)  
{  
    //corps de la boucle  
    Console.WriteLine(« La condition est  
vraie »);  
    i++;  
}
```





Exercice 14

- A partir de 2 afficher les nombres suivants de 3 en 3 jusqu'à ce que vous dépassiez 21.
- Chaque multiple doit être affiché sur une nouvelle ligne.
- Vous devez utiliser une boucle while pour cet exercice.




Exercice 15

- Demander à l'utilisateur de taper des nombres jusqu'à ce que ce dernier tape autre chose. Puis afficher la somme des nombres entrés par l'utilisateur.



Exercice 16

- 
- Taper un nombre n . Diviser ce nombre par 2 de manière successive jusqu'à ce que le résultat soit inférieure ou égale à 1. Afficher le nombre de division nécessaire pour atteindre ce résultat.

Types de référence

- Classes
 - Classe de base fondamentale de tous les autres types : `object`
 - Chaînes Unicode: `string` , qui représente une séquence d'unités de code UTF-16
 - Types définis par l'utilisateur de la forme `class C {...}`
 - Et plein d'autres classes du framework
- Interfaces
 - Types définis par l'utilisateur de la forme `interface I {...}`
- Collections
 - Unidimensionnel, multidimensionnel et en escalier. Par exemple : `int[]` , `int[,]` et `int[][]`
- Délégués
 - Types définis par l'utilisateur de la forme `delegate int D(...)`

Les collections

- Using System.Collections.Generic;
- [Dictionary<TKey,TValue>](#)
 - Représente une collection de paires clé/valeur organisées en fonction de la clé.
 -
- [List<T>](#)
 - Représente une liste d'objets accessibles par index. Fournit des méthodes de recherche, de tri et de modification de listes.
 -
- [Queue<T>](#)
 - Représente une collection d'objets premier entré, premier sorti (FIFO).
 -
- [SortedList<TKey,TValue>](#)
 - Représente une collection de paires clé/valeur triées par clé en fonction de l'implémentation [IComparer<T>](#) associée.
 -
- [Stack<T>](#)
 - Représente une collection d'objets dernier entré, premier sorti (LIFO).
 -
- [ArrayList](#)
 - Représente un tableau d'objets dont la taille est augmentée de manière dynamique selon les besoins.
- [Hashtable](#)
 - Représente une collection de paires clé/valeur qui sont organisées en fonction du code de hachage de la clé.

Tableaux

- Taille fixe définie quand le tableau est crée.
- Ne peut contenir que des objets d'un type particulier.
- Unidimensionnel :
`int[] array = new int[3];`
- Multidimensionnel :
`float[,] array = new float[5,4];`
- En escalier :
`bool[][] jagged =
{
 new int[]{1,5,6,3},
 new int[]{2,9,777},
 new int[]{0,3}
}`



Manipulation de tableaux

Initialisation :

- `int[] nomTableauInt = new int[6];`
- `string[] array = new string{ « test », « ça » };`

Assignation :

`nomTableauInt[2] = 66;`

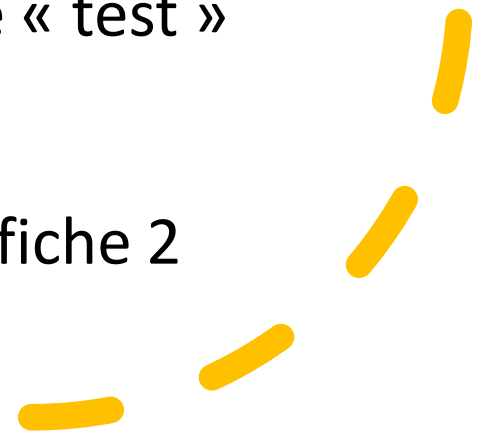
`//la 3e valeur du tableau est maintenant 66.`

Lecture :

`Console.WriteLine(array[0]);//affiche « test »`

Taille du tableau:

`Console.WriteLine(array.Length);//affiche 2`




foreach

```
int[] montableau = {0,1,2,3}

foreach(int i in montableau)
{
    //corps de la boucle
    Console.WriteLine(« Mon tableau
    contient l'élément {0} », i);
}

//Affichage
//0
//1
//2
//3
```





Exercice 17

- Écrire un programme qui remplit un tableau avec les chiffres entre 5 et 13.
- Remplacer le 3e élément du tableau par 111.
- Afficher tous les éléments du tableau




Exercice 18

- Ecrire une fonction qui remplace tous les nombre pair d'un tableau d'entiers par leur index dans le tableau.
- Afficher dans la console le tableau original et le tableau modifié.



Exercice 19

- 
- Ecrire une fonction qui crée un nouveau tableau à partir de 2 autres en prenant alternativement dans l'un et dans l'autre. Puis afficher le nouveau tableau dans la console.
 - Les tableaux d'origines n'ont pas forcément la même taille.

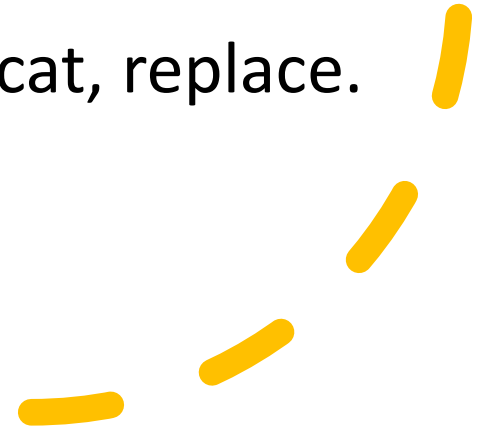


Exercice 20

- Écrire une fonction qui insère un tableau à un index précis dans un autre tableau.
- Exemple
array1 = {0,1,2,3};
array2 = {6,7,8};
array3 = F(array1,array2,2);
//array3
// {0,1,6,7,8,2,3}

string

- Ce sont des tableaux de caractères.
- Permet de représenter du texte.
- Peut-être coûteux à manipuler.
- Fonctions spécifiques : split, concat, replace.





Exercice 21

- Ecrire un programme qui remplit un tableau avec tous les caractères d'un mot tapé par l'utilisateur.
- Afficher le mot contenu dans le tableau en intercalant un « . » entre chaque lettre.
- Inverser le contenu du tableau puis afficher le mot obtenu dans la console.




Exercice 22

- Ecrire un programme qui remplace tous les caractères # rencontrés dans une string.
- Afficher la string modifiée dans la console.



Exercice 23

- Ecrire un programme qui annonce si une string est un palindrome ou pas.
- 

List<T>

- La taille s'adapte en fonction des besoins.
- Plusieurs méthodes pratiques de manipulations : Add, Remove, Insert, RemoveAt, Count
- Moins performant que les tableaux.





Exercice 24

- Créer un programme qui demande à l'utilisateur de taper des noms jusqu'à ce qu'il tape un nombre. L'utilisateur doit taper au minimum 6 noms.
- Stocker les noms dans une liste.
- Retirer le 2^e et le 5^e nom.
- Insérer « Toto » à la 3^e place.
- Inverser la liste puis afficher son contenu.

HashSet<T>

- Les valeurs sont forcément uniques.
- L'ordre des valeurs n'est pas garanti.
- La recherche est beaucoup plus rapide que dans des listes ou des tableaux.
- Obligatoire d'utiliser un foreach pour accéder aux valeurs





Exercice 25

1. Créer un hashset contenant des entiers.
2. Ajouter dans ce hashset tous les nombres en 0 et 1000000.
3. Faire pareil avec une liste.
4. Comparer le temps qu'il vous faut pour tester si la liste et le hashset contiennent le nombre 900000.

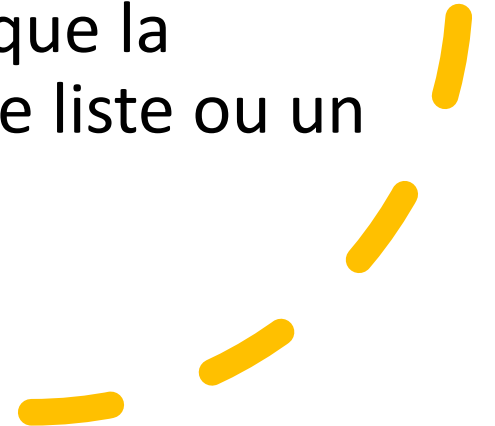


Exercice 26

1. Créer un hashset de int.
2. Ajouter dans ce hashset tous les nombres entre 0 et 100.
3. Ajouter une seconde fois le nombre 0.
4. Retirer tous les nombres entre 40 et 50.
5. Afficher dans une string le contenu du hashset.

Dictionary<T,U>

- Permet de lier des clés à des valeurs.
- Les clés sont forcément uniques.
- L'ordre des pairs clé/valeur n'est pas garanti.
- La recherche d'une clé dans un dictionnaire est beaucoup plus performante que la recherche d'un élément dans une liste ou un tableau.





Exercice 27

1. Créer un dictionnaire pour représenter des scores. La clé doit être une string et la valeur un int.
2. Remplir le dictionnaire avec 5 paires (= nom + score).
3. Mesurer le temps nécessaires pour tester la présence d'une clé particulière.
4. Remplir le dictionnaire avec 10000 paires aléatoires.
5. Mesurer le temps nécessaire pour trouver une des 5 premières paires.
6. Afficher le nom et le score de 5 paires choisies aléatoirement dans le dictionnaire.

Les classes

- Une classe est un ensemble de fonctions et de variables qui permet de définir le comportement d'un objet.
- Certains comportements comme la création ou la destruction sont automatiquement créés par le compilateur si le programmeur ne les définit pas.

2 références

```
public class Person
```

```
{
```

```
    public string Name;
```

```
    // Constructor that takes no arguments:
```

0 références

```
    public Person()
```

```
    {
```

```
        Name = "unknown";
```

```
    }
```

```
    // Constructor that takes one argument:
```

0 références

```
    public Person(string name)
```

```
    {
```

```
        this.Name = name;
```

```
    }
```

```
}
```

Constructeur

- Le constructeur est une fonction publique qui porte le même nom que la classe et n'a pas de type de retour.
- Le constructeur par défaut est un constructeur sans argument créé par le compilateur quand le programmeur n'en définit aucun.
- Quand le constructeur par défaut est utilisé les variables sont initialisées en utilisant la valeur par défaut de leur type.

2 références

```
public class Person
```

```
{
```

```
    public string Name;
```

```
    // Constructor that takes no arguments:
```

0 références

```
    public Person()
```

```
    {
```

```
        Name = "unknown";
```

```
    }
```

```
    // Constructor that takes one argument:
```

0 références

```
    public Person(string name)
```

```
    {
```

```
        this.Name = name;
```

```
    }
```

```
}
```

Finaliseur

- Le finaliseur (anciennement destructeur) est une fonction privée qui porte le même nom que la classe et n'a pas de type de retour. Il est précédé par un ~.
- En règle générale, le destructeur n'est jamais appelé explicitement.

```
//destructor  
0 références  
~Person()  
{  
    Console.WriteLine("Destroying {0}", Name);  
}
```

Variables globales

déclarée dans une classe mais en dehors d'une méthode.

Peut-être lu ou modifiée depuis n'importe quelle méthode de cette classe.

Si elle est publique, peut-être lu ou modifiée depuis l'extérieur de la classe.

Variables locales


déclarée dans un bloc de code (méthode, if, boucle ...).

N'existe que dans le bloc de code dans lequel elle est déclarée.

Il est impossible de lui donner un accesseur.



Exercice 28



Créer une classe permettant de représenter un élève. Cette classe doit permettre de connaître le nom, le prénom, la date de naissance, les moyennes de l'élève dans chaque matière (vous pouvez en inventer 5) et sa moyenne générale.



Exercice 29

Créer une classe permettant de représenter une classe, une classe (= un groupe d'élève). Une classe a un nom, un niveau et elle est composée d'élèves. Chaque classe a un nombre d'élèves maximum.

Une classe a une méthode promotion qui retourne tous les élèves ayant plus de 10 en moyenne générale.



Exercice 30

Créer une classe école qui permet à un élève de rentrer en classe de niveau 0 et de sortir diplômé en classe de niveau 6.

L'école est composée de 3 classes de chaque niveau. Elle gère l'intégration dans une classe de niveau supérieur (ou la remise du diplôme) pour les élèves promus.

Au cours de chaque promotion l'école remet une récompense spéciale au promu ayant la meilleure moyenne générale dans chaque niveau.



Exercice 31

Créer un programme qui rempli les conditions suivantes:

- Créer une école.
- Remplir chaque classe avec des élèves créés de manières aléatoires.
- Simuler le passages de 6 ans (= 6 promotions) en continuant à remplir les classes de plus bas niveau.
- Les notes des élèves sont aléatoires.
- Afficher les élèves promus et les élèves récompensés à chaque promotion.
- Aucune classe ne peut dépasser son effectif maximum. Mettez en place une solution pour gérer les cas particuliers où cette situation pourrait arriver.