



Variable





Qu'est-ce qu'une variable ?

Une variable est une zone mémoire dans laquelle on pourra stocker une valeur.

Elle est composée de 3 éléments :

- un nom
- un type
- une valeur



Déclaration d'une variable

Pour déclarer une variable on associe un type à un nom, avant d'associer celui-ci à une valeur.

Pour être capable de bien associer une variable à une valeur, il faut déjà comprendre les types.

Exemple :

```
string name = "Victor"
```

string étant le type

name étant ma variable

Victor étant la valeur à laquelle ma variable est associer



Type



Qu'est-ce qu'un type ?

Un type est une façon de classer les variables. Il existe plusieurs types de variables :

- 📌 `int` : nombre entier
- 📌 `float` : nombre à virgule
- 📌 `string` : chaîne de caractères
- 📌 `bool` : booléen (vrai ou faux)



Pourquoi utiliser des types ?

Les types permettent de classer les variables, et de les utiliser plus facilement.

Par exemple, si on a une variable de type `int`, on pourra l'utiliser pour stocker des nombres entiers. Si on a une variable de type `string`, on pourra l'utiliser pour stocker des chaînes de caractères.

Si on demande à l'utilisateur d'entrer un texte, on va stocker cette valeur dans une variable de type `string` et non `int`.



Et par exemple, on ne peut pas additionner une chaîne de caractères avec un nombre.



Int

Le type `int` est le type pour les nombres entiers. Un nombre entier est un nombre qui ne contient pas de virgule. Il peut aussi être négatif.

Exemple : `125` , `0` , `-12`



Float

Le type `float` est le type pour les nombres à virgule.

Un nombre à virgule est un nombre qui contient une virgule. Il peut aussi être négatif.

Exemple : `1.025` , `0.0` , `-12.256`



String

Le type `string` est le type pour les chaînes de caractères.

Une chaîne de caractères est une suite de caractères qui forme un mot ou une phrase.

Exemple : `"hello world"`, `"hello"`, `"world"`



Bool

Le type `bool` est le type pour les valeurs booléennes.

Une valeur booléenne est une valeur qui est vraie ou fausse.

Exemple : `true` , `false`



Exemple de déclaration

```
int nom_de_variable = 12;  
float variable_floating_point = 2.0f;  
string what_ever_name = "bonjour les amiches";  
bool name_true_false = true;
```

Il faut bien comprendre ici que le nom des variables est libre de choix.

Mais pour autant il ne faut pas mettre n'importe quoi, car celle ci participe à la clarté de votre code.



Bien choisir ses noms de variable

Vous demandez à un utilisateur de mettre le nom d'une voiture en input

Vous stockerez cette valeur dans une string et vous la nommerez en rapport avec l'objet de votre demande,

Une voiture:

```
string marque_voiture;  
string car_brand;  
string brand_car;  
string car_name;
```



Exercice sur les variables

- 📌 Creer une variable `age` de type `int` et lui assigner une valeur.
- 📌 Creer une variable `name` de type `string` et lui assigner une valeur.
- 📌 Creer une variable `isAdult` de type `bool` et lui assigner une valeur.
- 📌 Creer une variable `height` de type `float` et lui assigner une valeur.
- 📌 Afficher la valeur de chaque variable avec la fonction
`Console.WriteLine(nom_de_variable)`



Exercice solution

```
int age = 20;  
string name = "Jean";  
bool isAdult = true;  
float height = 1.83f;  
  
Console.WriteLine(age);  
Console.WriteLine(name);  
Console.WriteLine(isAdult);  
Console.WriteLine(height);
```



Exercice sur le nommage des variables

- 📌 Vous possédez un café et vous voulez automatiser la visualisation de certains chiffres. Pour cela vous demandez au serveur du jour de rentrer le nombre de café servit, la quantité en graine (en gramme) utilisée, le nombre de client servit et le nom du serveur, pour un programme que vous avez confectionné.
Créez des variables adaptées à chacune de ses informations et nommez judicieusement.



Exercice

📌 Bob, Richard et Mark sont pêcheurs sur le même bateau. Ils font la compétition du nombre de poisson péchés et du poids total péchés. Le capitaine leur propose de créer un petit programme dans lequel chacun rentrera leurs noms, suivit du nombre de poisson péchés aujourd'hui ainsi que le poids total de leurs prise du jour.
Créer deux variables pour chacun d'eux et nommez les.



Lecture

Une fois ma variable déclaré, elle pourra être utilisée tout au long de mon programme. Pour cela nous utiliserons son nom sans avoir à redefinir son type.



Comment utiliser ma variable

Pour utiliser une variable, c'est à dire accéder à la valeur stockée dans celle-ci, il suffit de l'appeler par son nom.

Pour autant, nous ne pouvons pas rééefinir son type après déclaration.

```
int nom_de_variable;  
  
nom_de_variable = 153;  
  
Console.WriteLine(nom_de_variable); // affichera 153  
// nom_de_variable = "bonjour" --> erreur car "bonjour" est une string et non un int
```



Attribution

Pour affecter une valeur à une variable, nous utiliserons l'opérateur
=.

Cet opérateur permet d'affecter une valeur à une variable.



Comment attribuer une valeur à ma variable

Il est possible de redéfinir la valeur d'une variable en lui attribuant une nouvelle valeur.

```
int nom_de_variable;  
  
nom_de_variable = 153;  
Console.WriteLine(nom_de_variable); // affichera 153  
  
nom_de_variable = 12;  
Console.WriteLine(nom_de_variable); // affichera 12
```



Les opérateurs

Les opérateurs sont des symboles qui permettent d'effectuer des opérations sur des variables ou des valeurs.

Nous avons vu précédemment l'opérateur d'assignation `=` pour associer une variable à une valeur.

Mais il en existe pour faire des opérations arithmétiques, pour comparer des variables, pour en incrémenter et pour concaténer.



Les opérations arithmétiques

Il est possible d'effectuer des opérations sur les variables.



Addition

Pour additionner deux variables, nous utiliserons l'opérateur `+`.

```
int a = 3;  
int b = 5;  
  
int c = a + b;  
Console.WriteLine(c); // affichera 8
```



Soustraction

Pour soustraire deux variables, nous utiliserons l'opérateur `-`.

```
int a = 3;  
int b = 5;  
  
int c = a - b;  
Console.WriteLine(c); // affichera -2
```



Multiplication

Pour multiplier deux variables, nous utiliserons l'opérateur `*`.

```
int a = 3;  
int b = 5;  
  
int c = a * b;  
Console.WriteLine(c); // affichera 15
```



Division

Pour diviser deux variables, nous utiliserons l'opérateur `/`.

```
int a = 3;  
int b = 5;  
  
int c = a / b;  
Console.WriteLine(c); // affichera 0
```



Modulo

Pour calculer le reste d'une division entre deux variables, nous utiliserons l'opérateur `%`.

```
int a = 31;  
int b = 5;  
  
int c = a % b;  
Console.WriteLine(c); // affichera 1
```



Exercice

- 📌 Creer une variable pizza (entier) qui sera egale a 5.
- 📌 Creer une variable part_de_pizza (entier) qui egale a 6.
- 📌 Creer une variable nbr_part_de_pizza qui sera egale au nombre de pizza multiplier par part_de_pizza.



Exercice

Vous etes 9 Personnes.

- 📌 Creer une variable nbr_personne qui sera egale a 9.
- 📌 Creer une variable part_de_pizza_par_personne, qui sera egale au mon de part_de_pizza diviser par le nbr_personne.
- 📌 Creer une variable part_de_pizza_restante qui sera egale au nombre de par restante apres distribution (utiliser modulo)



Solution

```
int pizza = 5;  
  
int part_de_pizza = 6;  
  
int nbr_part_de_pizza = pizza * part_de_pizza;  
  
int nbr_personne = 9;  
  
int part_de_pizza_par_personne = nbr_part_de_pizza / nbr_personne;  
  
int part_de_pizza_restante = nbr_part_de_pizza % nbr_personne;
```



Opérations de comparaisons

La comparaison de variable est au cœur des boucles logiques et conditionnelles, elle vous permettra de comparer deux variables.

Le retour d'une comparaison est une variable `bool` true ou false en fonction de la vérité de comparaison.



Egalité

Pour comparer si deux variables sont égales, nous utiliserons l'opérateur `==`.

```
int a = 3;  
int b = 5;  
  
bool c = a == b;  
Console.WriteLine(c); // affichera false
```



Superieur

Pour comparer si une variable est supérieure à une autre, nous utiliserons l'opérateur `>`.

```
int a = 3;  
int b = 5;  
  
bool c = a > b;  
Console.WriteLine(c); // affichera false
```



Inferieur

Pour comparer si une variable est inférieur à une autre, nous utiliserons l'opérateur `<`.

```
int a = 3;  
int b = 5;  
  
bool c = a < b;  
Console.WriteLine(c); // affichera true
```



Supérieur ou égal

Pour comparer si une variable est supérieure ou égale à une autre, nous utiliserons l'opérateur `>=`.

```
int a = 3;  
int b = 5;  
  
bool c = a >= b;  
Console.WriteLine(c); // affichera false
```



Inférieur ou égal

Pour comparer si une variable est inférieure ou égale à une autre, nous utiliserons l'opérateur `<=`.

```
int a = 3;  
int b = 5;  
  
bool c = a <= b;  
Console.WriteLine(c); // affichera true
```



Exercice

Creer 4 variables, `a`, `b`, `c` et `d` de type `float` telque :

- 📌 `a = 1.5`
- 📌 `b = 5`
- 📌 `c = 1.5`
- 📌 `e = 0.5`



Exercice

En utilisant l'operateur `>` et 2 des 4 variables (`a`, `b`, `c`, `d`);
Creer 2 variable de type `bool` telque :

📌 `f = true`

📌 `g = false`



Exercice

En utilisant l'operateur < et 2 des 4 variables (a , b , c , d);
Creer 2 variable de type bool telque :

📌 h = true

📌 i = false



Exercice

En utilisant l'operateur `==` et 2 des 4 variables (`a`, `b`, `c`, `d`);
Creer 2 variable de type `bool` telque :

📌 `j = true`

📌 `k = false`



Solution :

```
float a = 1.5f;  
float b = 5f;  
float c = 1.5f;  
float d = 0.5f;  
  
bool f = b > a;  
bool g = d > a;  
  
bool h = c < b;  
bool i = c < d;  
  
bool j = a == c;  
bool k = d == c;
```



Opérateurs d'incrémentation

En informatique, l'incrémantation est l'opération qui consiste à ajouter 1 à un compteur.

Pour incrémenter une variable de 1, nous utiliserons l'opérateur `++`.

```
int a = 3;  
  
a++;  
Console.WriteLine(a); // affichera 4
```



Operateurs de decrementation

L'opération inverse, la décrémentation, consiste à retirer 1 au compteur

Pour décrementer une variable de 1, nous utiliserons l'opérateur `--`.

```
int a = 3;  
  
a--;  
Console.WriteLine(a); // affichera 2
```



Opérateur incrémantation de n

Pour incrémenter une variable de n, nous utiliserons l'opérateur `+=`.

```
int a = 3;  
  
a += 5;  
Console.WriteLine(a); // affichera 8
```



Opérateur décrementation de n

Pour décrementer une variable de n, nous utiliserons l'opérateur `-=`.

```
int a = 3;  
  
a -= 5;  
Console.WriteLine(a); // affichera -2
```



Opérateur décrementation de n

Créer deux variables `int : salaire` et `solde_banque`

- 📌 Initialiser `salaire` à 1280 et `solde_banque` à 329.
- 📌 Incrementer `solde_banque` de `salaire`



Opérateur décrementation de n

Créer deux nouvelles variables `int : depense_course` et `loyer`

- 📌 Initialiser `depense_course` à 450
- 📌 Initialiser `loyer` à 600
- 📌 Décrementer `soldé_banque` de `depense_course`
- 📌 Décrementer `soldé_banque` de `loyer`



Opérateur décrementation de n

Vous avez une augmentation de salaire de 150

📌 Incrementer votre salaire de 150



Solution :

```
int salaire = 1280;
int solde_banque = 329;

solde_banque += salaire;

int depense_course = 450;
int loyer = 600;

solde_banque -= depense_course;
solde_banque -= loyer;

salaire += 150;
```



Opérateurs de concatenation

Pour concatener deux chaînes de caractères, nous utiliserons l'opérateur `+`.

```
string a = "Hello";
string b = "World";

string c = a + b;
Console.WriteLine(c); // affichera HelloWorld
```



Scope

Le Scope | La portée

```
{  
}  
}
```



Qu'est-ce que le scope

Le scope est la portée d'une variable, portée géographique : c'est là où la variable est visible dans le code.

Les variables déclarées dans une :

- 📌 boucle ne peuvent être utilisées que dans cette boucle.
- 📌 fonction ne peuvent être utilisées que dans cette fonction.



Exemple:

```
int a = 3;  
{  
int b = 5;  
Console.WriteLine(a); // affichera 3  
Console.WriteLine(b); // affichera 5  
}  
Console.WriteLine(b); // affichera une erreur
```

Ici, `b` est déclaré dans des accolades `{ }`, et n'est donc accessible qu'au sein de ces `{ }`. C'est pour ça que nous avons une erreur au second `Console.WriteLine(b)`



Variable globale

Une variable globale est une variable déclarée en dehors de tout scope. Elle est donc accessible partout dans le code.

Dans l'exemple ci dessus, `a` est une variable globale car elle est déclarée en dehors de tout scope (`{ }`), et est accessible partout (toujours ci dessus nous avons accès à `a` dans le même scope où est déclaré `b`)



Variable locale

Une variable locale est une variable déclarée dans un scope, et qui ne peut être utilisée que dans ce scope.

b est une variable dite locale à son scope.

```
int a = 3;
{
    int b = 5;
    Console.WriteLine(a); // affichera 3
    Console.WriteLine(b); // affichera 5
}
Console.WriteLine(b); // affichera une erreur
```



Scope dans un scope

```
{      // debut du premier scope
int a = 3;
Console.WriteLine(a);
{
    // debut du scope 2
    int b = 4;
    Console.WriteLine(a);
    Console.WriteLine(b);
    {
        // debut du scope 3
        int c = 6;
        Console.WriteLine(a);
        Console.WriteLine(b);
        Console.WriteLine(c);

    }      // fin du scope 3
            // variable c n'est plus accessible ici !!
}  // fin du scope 2
            // variable b n'est plus accessible ici !!
}      // fin du premier scope
```



Scope dans un scope

Une variable est définie dans son scope comme vu précédemment et ce scope possèdera toutes les entités présente dans celle-ci.

C'est pour ca que le programme ci dessus fonctionne et affiche bien les résultats attendus



Exercice

Creer 3 variable globals telque :

- 📌 **jean** une string initialiser a "Jean"
- 📌 **ville** une string initialiser a "Paris"
- 📌 **annee** un int initialiser a 1998

Exercice

Creer un scope dans lequel sera instancier :

- 📌 `annee2` un int egale a 1999
- 📌 `lea` une string egale a "Lea"

Creer un second scope dans le premier et instancier :

- 📌 `fact_check` un booleen qui verifiera `annee < annee2` ;
- 📌 une string `copie_ville` qui sera egale a `ville`



Solution :

```
string jean = "Jean";
string ville = "Paris";
int annee = 1998;
{
int annee2 = 1999;
string lea = "Lea";
{
    bool fact_check = annee < annee2;
    string copie_ville = ville;
}
}
```



[] : Tableau



Definition

Un tableau est une collection d'entités du même type. Chaque entité est accessible via un index.

Un tableau peut être visualisé comme un coffre qui contient d'autres coffres.



Comment déclarer un tableau

Pour déclarer un tableau, nous utiliserons la syntaxe suivante:

```
type[ ] nomTableau = new type[taille];
```

2 nouveaux éléments ici :

📌 []

📌 new

taille correspond au nombre d'éléments dans mon tableau



Exemple de déclaration de tableaux:

```
int[] tableau_int = new int[10]; // tableau de 10 entiers
```

```
string[] tableau_string = new string[12]; // tableau de 12 string
```



Remplir un tableau

Pour remplir un tableau, nous utiliserons la syntaxe suivante:

```
nomTableau[index] = valeur;
```

Le premier index de mon tableau est 0.



Exemple de remplissage de tableau:

```
int[] tableau_int = new int[4]; // tableau de 4 entiers  
  
tableau_int[0] = 3; //index = 0, valeur = 3;  
tableau_int[1] = 5;  
tableau_int[2] = 7;  
tableau_int[3] = 9;
```



Initialisation lors de la déclaration

Vous pouvez éviter l'expression `new` et le type de tableau lorsque vous initialisez un tableau lors de la déclaration, comme indiqué dans le code suivant.

```
int[] array2 = { 1, 3, 5, 7, 9 };  
string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```



Exercice :

- 📌 Creer un tableau de string `mois` pour y mettre tout les mois de l'annee et instancier le.



Solution :

```
string[] mois = {"janvier", "fevrier", "mars", "avril", "mai", "juin", "juillet", "aout", "septembre", "octobre", "novembre", "decembre" };  
--- reponce alternative  
  
string[] mois = new string [12];  
mois[0] = "janvier";  
. . .  
mois[11] = "decembre";
```





Accéder à la valeur dans le tableau

Pour accéder à la valeur dans le tableau, nous utiliserons la syntaxe suivante:

```
nomTableau[index]
```

!! Le premier index de mon tableau est 0.

Exemple d'accès à la valeur dans un tableau:

```
int[] tableau_int = new int[4]; // tableau de 4 entiers

tableau_int[0] = 3; //index = 0, valeur = 3;
tableau_int[1] = 5;
tableau_int[2] = 7;
tableau_int[3] = 9;

Console.WriteLine(tableau_int[0]); // affichera 3
Console.WriteLine(tableau_int[1]); // affichera 5
Console.WriteLine(tableau_int[2]); // affichera 7
Console.WriteLine(tableau_int[3]); // affichera 9
```



Exercie

Avec l'exercice precedent,

Creer un tableau de string `mois_avec_e`, qui contiendra tout les mois qui possede un `e` dans leurs nom.

Celui-ci sera initialiser seulement a l'aide d'index et le tableau de `mois`.



Solution :

```
string[] mois = {"janvier", "fevrier", "mars", "avril", "mai", "juin", "juillet", "aout", "septembre", "octobre", "novembre", "decembre" };

string[] mois_avec_e = new string [7];
mois_avec_e[0] = mois[0];
mois_avec_e[1] = mois[1];
mois_avec_e[2] = mois[6];
mois_avec_e[3] = mois[8];
mois_avec_e[4] = mois[9];
mois_avec_e[5] = mois[10];
mois_avec_e[6] = mois[11];
```



Méthode de base

Un tableau en C# est un object qui possède des certaines propriétés déjà intégré.

Pour les tableaux, la principale et la plus utilisée est :



`Length` : retourne la taille du tableau

Principalement utilisée pour parcourir votre tableau.



Exemple

```
int[] tableau_int = new int[4]; // tableau de 4 entiers  
  
tableau_int[0] = 3; // index commence a 0  
tableau_int[1] = 5;  
tableau_int[2] = 7;  
tableau_int[3] = 9;  
  
Console.WriteLine(tableau_int.Length); // affichera 4
```





Logique



Condition

Une condition est une expression qui retourne un booleen (true ou false) .

En fonction de la valeur de la condition, un bloc de code sera exécuté (true) ou non (false).

Tous les opérateurs de Comparaison peuvent être utilisé dans les conditions

Opérateur de comparaison	Signification
<code>==</code>	égal à
<code>!=</code>	différent de
<code>></code>	supérieur à
<code>>=</code>	supérieur ou égal à
<code><</code>	inférieur à
<code><=</code>	inférieur ou égal à



If

La condition la plus simple est le `if` qui permet d'exécuter un bloc de code si la condition est vraie.

```
if (condition 1)
{
    // code a executer si la condition 1 est vrai
}
```



Exercice :

Creer deux variable `int` `soldé_camille` et `soldé_bob`
initialiser les telque :

```
soldé_camille = 1111;  
soldé_bob = 289;
```

Creer une condition `if` si `soldé_camille` est supérieur au `soldé_bob` et
y mettre le code suivant

```
Console.WriteLine("camille a un solde supérieur à bob " + soldé_camille);
```

Executer le code.



Solution :

```
int solde_camille = 1111;  
int solde_bob = 289;  
  
if (solde_camille > solde_bob) {  
  
    Console.WriteLine("camille a un solde superieur a bob " + solde_camille);  
}
```



Else

La condition `else` permet d'exécuter un bloc de code si la condition est fausse.

```
if (condition 1)
{
    // code à exécuter si la condition 1 est vraie
}
else
{
    // sinon, code à exécuter si la condition 1 est fausse
}
```



Exercice :

Reprendre le code précédent et ajouter au scope de else :

```
Console.WriteLine("Bob a un solde superieur a bob " + solde_bob);
```

Executer le code.

Changer le solde de bob a 2222;

Executer le code.

Quelle chose a change dans l'affichage du terminal?



Else if

La condition `else if` permet d'exécuter un bloc de code si la condition est fausse et que l'on souhaite vérifier une autre condition.

```
if (condition 1)
{
    // code à exécuter si la condition 1 est vraie
}
else if (condition 2)
{
    // sinon, code à exécuter si la condition 1 est fausse et que condition 2 est vraie
}
else
{
    // code à exécuter si la condition 1 est fausse et que condition 2 est fausse
}
```



Exemple

```
int a = 5;
int b = 10;

if (a > b)
{
    Console.WriteLine("a est supérieur à b");
}
else if (a < b)
{
    Console.WriteLine("a est inférieur à b");
}
else
{
    Console.WriteLine("a est égal à b");
}
```

Exécuter le code.

```
Console.WriteLine("a est égal à b");
```



Exercice

```
Changer int a = 120;  
Executer le code.  
Changer int a = 120;  
Changer int b = 120;  
Executer le code.
```



Exercice

```
int ton_age = mettre ton age;  
int nombre_enfant = mettre ton nombre d enfant;
```

1 Créer une condition pour vérifier si tu es majeur.

Elle affichera \ (avec Console.WriteLine\) si tu es majeur, ou si tu es mineur

2 Créer une ou plusieurs conditions pour vérifier si \:

tu n as pas d enfant, // afficher je n'est pas d'enfant

3 tu as moins de 3 enfants mais tu as des enfants, // afficher j'ai nbr d enfant

4 tu as 4 ou moins mais plus que 2, // j'ai plus que 2 enfants mais ce n'est pas une famille nombreuse

5 tu as plus que 4 enfants, // j'ai une famille nombreuse

```
int ton_age = mettre ton age;
int nombre_denfant = mettre ton nombre d enfant;

if (ton_age >= 18)
{
Console.WriteLine("Tu es majeur");
}
else
{
Console.WriteLine("Tu es mineur");
}
if (nombre_denfant == 0) // condition 1
{
Console.WriteLine("Je n'ai pas d'enfant");
}
else if (nombre_denfant < 3) // si condition 1 fausse, condition 2
{
// autre possibilité, nbr_denfant <= 2
Console.WriteLine("J'ai " + nombre_denfant + " enfant");
}
```



Reponse

```
else if (nombre_enfant <= 4) // condition 3
{
    Console.WriteLine("J'ai plus que 2 enfants mais ce n'est pas une famille nombreuse");
}
else // condition 4 , si aucune des conditons n'étaient bonne
{
    Console.WriteLine("J'ai une famille nombreuse");
}
```



Opérateur Booleen

Les opérateurs booleen permettent de combiner des conditions.

Opérateur	Signification
&&	ET
	OU
!	NON



ET

L'opérateur ET permet d'executer un bloc de code si les deux conditions sont vraies.

```
if (condition 1 && condition 2)
{
    // code à executer si les deux conditions sont vraies
}
```



OU

L'opérateur OU permet d'executer un bloc de code si une des deux conditions sont vraies.

```
if (condition 1 || condition 2)
{
    // code à executer si une des deux conditions sont vraies
}
```



NON

L'opérateur NON permet d'inverser la valeur de la condition.

```
if (!condition)
{
    // code à executer si la condition est fausse
}
```



Exercice

```
int nbr_pizza = 14;  
int nbr_couvert = 30;  
int nbr_personne = 12;
```

1 Creer une condition qui verifie :

- tout le monde aura 2 couverts au moins
ET
- tout le monde aura 1 pizza au moins

2 Creer une condition qui verifie :

- tout le monde aura 2 couvert au moins
OU
- tout le monde aura 1 pizza au moins



Priorité des parenthèses

Les opérateurs booleen sont évalués de gauche à droite dans une condition.

Les parenthèses `()` fonctionne en informatique comme en mathématiques

La priorité de la condition se dirige dans la parenthèse en 1er.

Table de verite

A	B	A && B
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux



A	B	A B
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux



A	!A
Vrai	Faux
Faux	Vrai



Exercice

Reprendre l'exercice précédent en ajoutant la dernière condition :

3 Créer une condition qui vérifie :

- tout le monde aura 2 couverts au moins
ET
- tout le monde aura 1 pizza au moins
ET
 - il y a plus de 20 couverts
OU
 - il y a plus de 10 personnes



Boucle

Une boucle permet d'exécuter plusieurs fois un bloc de code, tant que la condition est vraie.



Boucle while

La boucle while permet de répéter un bloc de code tant qu'une condition est vraie.

```
while (condition) // tant que la condition est vraie
{
    // bloc de code à repéter
}
```



Exemple pratique

Les boucles `while` sont très pratique pour, un exemple: itérer sur un tableau.

Rappel,

Un tableau possède plusieurs éléments d'un même type;

Un tableau possède une fonction qui permet de connaître sa longueur : `.Length` ;

```
// parcourir son tableau avec un boucle while
int[] tableau = { 1, 2, 3, 4, 5 };

int index = 0;
while (index < tableau.Length) // tant que index est plus petit que la longeur du tableau
{
    Console.WriteLine(tableau[i]); // afficher l'element i du tableau
    index++; // augmenter index de 1
}
```

A partir de maintenant nous utiliserons une convention de code pour nommer une variable `index` : `i`



Boucle foreach

La boucle for each permet de parcourir un tableau sans avoir besoin de connaitre sa longeur.

```
foreach (type element in tableau)
{
    // bloc de code a repeter
}
```



Boucle foreach

Mots clefs :

📌 **foreach**

📌 **in**

```
// parcourir son tableau avec un boucle while
int[] tableau = { 1, 2, 3, 4, 5 };

foreach (int elems in arr)
{
    Console.WriteLine(tableau[elems]); // afficher l'element elems du tableau
}
```

On voit bien ici, contrairement à la boucle `while`, `elems` s'incrémente automatiquement à la fin du bloc conditionnelle.



Fonction

Les fonctions sont des blocs de code qui peuvent etre appellé n'importe quand dans le code.

Une fonction est une serie d'instruction qui permet de realiser une tache precise.

Elle peut recevoir un ou des **parametres** en entree et peut renvoyer une variable en sortie, le **retour**.



Visualiser une fonction

Une fonction peut etre vu comme une usine avec tapis roulant d'entre (parametres) et tapis roulant de sortis (retour).

On donne un ou des matiere premiere a l'usine en entree (parametres), et l'usine nous resortira un produit fini (retour).



Block de code, Organiser son code pour plus de clarete

Une fonction permet d'organiser son code pour plus de clareté, de separer les taches en bloc precie.

```
void Main()
{
    // code principal
}
void fonction1()
{
    // code de la fonction 1
}
```



Scope de fonction

Les variables créées dans une fonction ne sont pas visibles dans les autres fonctions.

La fonction va déclarer son scope, donc son champ d'action.

```
void Main()
{
int a = 1;
fonction1();
Console.WriteLine("var in main " + a); // affiche 1
}
void fonction1()
{
int a = 2;
Console.WriteLine("var in function " + a);
}
```



Paramètres passe

Les paramètres sont l'entrée d'une fonction, ils permettent de passer des variables à la fonction.

```
void main()
{
int a = 3;
int b = 8;
function1(a, b);
}
void fonction1(int a, int b)
{
Console.WriteLine(a + b); // affiche 11;
}
```



Paramètres passe

Le nom des paramètres n'est pas obligatoirement identique à celle passer dans la fonction.

Ils seront, dans la fonction, générique pour pouvoir être claire.

Le Type par contre doit être le même.

```
void main()
{
int a = 3;

square(a);
print(a);
}

void square(int number_to_square)
{
Console.WriteLine(a * a); // affichera 9;
}

void print(string str) // error, parametre string, parametre passe int
{
Console.WriteLine(str); // affichera la string;
}
```



Parametre par default

Les parametres peuvent avoir une valeur par default, si il n'est pas passer a la fonction.

```
void main()
{
int a = 3;
int b = 8;
function1(a, b); // affichera 11
function1(a);    // affichera 5
}
void function1(int a, int b = 2)
{
Console.WriteLine(a + b);
}
```



Retour fonction

Une fonction peut renvoyer un résultat, ce résultat est un **type** de variable.

Ce **type** est placé devant le nom de la fonction, et celle-ci ne pourra renvoyer que ce **type**.

On utilisera le mot clé **return** suivit de la variable que l'on voudra retourner.

```
return_type          fonction_name( parametre, ... ) {  
    return variable_name ;  
}
```

Si une fonction ne renvoie rien, on dit qu'elle est de type **void**.

Exemple :

```
void main()
{
int a = 3;
int b = 8;
Console.WriteLine(square(a)); // affichera 9
Console.WriteLine(square(b)); // affichera 64

}
int square(int x)
{
return (x * x);
}
```



Associer le retour d'une fonction

On peut associer le retour d'une fonction à une variable.

```
void main()
{
int a = 3;
int b = 8;
int x = square(a);
int y = square(b);
Console.WriteLine(x); // affichera 9
Console.WriteLine(y); // affichera 64
}
int square(int x)
{
return (x * x);
}
```



Exercice

- 📌 Creer une fonction qui prend en parametre un tableau d'entier, et qui renvoie le plus grand element du tableau.
- 📌 Creer une fonction qui prend en parametre un tableau d'entier, et qui renvoie la somme de tous les elements du tableau.



Reference



Qu'est ce qu'une reference?

Une reference permet d'accéder à la variable et la modifier en dehors de son scope dans des fonctions.



Quand utiliser la reference de ma variable

La reference est utile dans des fonctons qui ont pour but de modifier les variables en parametre.

En C# certaines variable comme les `tableaux` sont de type reference de base.

Les variables classiques, `int` , `float` , `string` ne le sont pas.

```
main();

void function1(int[] arr) {
arr[0] = 100;
};

void function2(int num) {
num = 10;
};

void main() {

int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int num = 18;

Console.WriteLine("before go to fonction1 " + arr[0]); // affiche 1
Console.WriteLine("before go to fonction2 " + num); // affiche 18

function1(arr);
function2(num);

Console.WriteLine("after fonction1 "+ arr[0]); // affiche 100
Console.WriteLine("after fonction2 "+ num); // afficher 18

};
```



Comment utiliser la reference

Mot clef `ref` devant le type de la variable dans les parametre de la fonction.

```
main();

void function1(ref int num) { // mot clef ref
    num = 1000;
}

void main() {
    int num = 18;

    Console.WriteLine("before go to fonction2 " + num); // affichera 18
    function1(ref num); // mot clef ref
    Console.WriteLine("after fonction2 "+ num); // affichera 1000
}
```



Exercice avec les references

Ecrire une fonction `is_Billi` qui prend en parametre une reference sur string, renvoie un boolen.

La fonction verifie si l'argument est egale a `Billi` et renverra `true` si vrai.

Sinon modifier l'argument pour le transforme en `Billi` et renvoyer false.



Librairie

Une librairie est un ensemble de fonction déjà écrite, utilisable dans notre programme, réunie dans une classe.

Tout au long de ce cours nous avons utilisé `Console.WriteLine`,
Console étant une classe avec des fonctions comme WriteLine.



Random

La classe `Random` permet de generer des nombres aleatoires.

```
Random rnd = new Random();
int x = rnd.Next(0, 100); // x est un nombre aleatoire entre 0 et 100
```



Dictionnaire

La classe `Dictionary` permet de créer un tableau de `clef / valeur`.
Les avantages sont :

- 📌 les méthodes de base à disposition.
- 📌 Le faites que le "tableau" soit extensible.
- 📌 La recherche des éléments rapide.



Declaration

Un dictionnaire ce declare avec 2 types;

Un type pour la **clef**, un pour la **valeur**.

```
Dictionary<int , string> dict = new Dictionary<string, int>(); // int --> clef , string --> valeur
```



Clef Valeur

Un dictionnaire associe une clef a une valeur.

Une clef est unique.

Pour rechercher une valeur, nous utiliserons sa clef comme index.

```
Dictionary<int, string> dictionnaire = new Dictionary<int, string>();
dictionnaire.Add(1, "un");
dictionnaire.Add(2, "deux");
dictionnaire.Add(3, "trois");

Console.WriteLine(dictionnaire[1]); // affichera un
Console.WriteLine(dictionnaire[2]); // affichera deux
Console.WriteLine(dictionnaire[3]); // affichera trois
```



Fonction recherche

La fonction `ContainsKey` permet de rechercher si une clef existe dans le dictionnaire.

```
Dictionary<int, string> dictionnaire = new Dictionary<int, string>();
dictionnaire.Add(1, "un");
dictionnaire.Add(2, "deux");
dictionnaire.Add(3, "trois");

if (dictionnaire.ContainsKey(1)) {
    Console.WriteLine("La clef 1 existe dans le dictionnaire");
}
```

Methode Dictionary

- 📌 **Add** : permet d'ajouter une clef / valeur
- 📌 **Remove** : permet de supprimer une clef / valeur
- 📌 **Clear** : permet de vider le dictionnaire
- 📌 **ContainsKey** : permet de verifier si une clef existe
- 📌 **ContainsValue** : permet de verifier si une valeur existe





Qu'est ce que la POO (Programmation orientée object)

La programmation orienté objet est une méthode de programmation qui permet de structurer un programme en **objet**.

Un objet est une représentation d'une entité, comme une voiture, un chat, un humain, un programme, etc...



Pourquoi POO

La POO permet de créer des objets qui ont des propriétés et des méthodes.

Cela permet :

- 📌 Reutilisation du code
- 📌 Efficacité de l'appel de fonction
- 📌 Facilite la maintenance



Pourquoi POO

- 📌 Facilite le dépannage
- 📌 Facilite la lecture du code
- 📌 Facilite la modification du code



Les classes



Les classes

Une classe est une structure de données qui permet de définir un nouvel objet.

Une classe est un modèle de données.

Exemple : `Console` , `Dictionary` , `Math`



Declaration d'une classe

Une classe se declare avec le mot clef `class` suivi du nom de la classe.

```
public class Personne { // Declaration d'une classe Personne  
// ...  
}
```



Classes privees et publiques

Nous pouvons declarer les classes de deux manieres differentes, en utilisant les mots cles a mettre devant `class` :

 Public

 Private



Classe Private

Une classe private ne peut etre utiliser que dans la classe ou elle est declaree. Nous parlons ici de l'encapsulation d'une classe.

Car oui, une classe peut etre declarer dans une classe.

```
private class Personne { // Classe private  
// ...  
}
```



Classe Public

Une classe public peut etre utilisee dans toute le programme.

```
public class Personne { // Classe public  
// ...  
}
```



Attributs

Les attributs sont les variables d'une classe.

Elles peuvent elles aussi etre `public` ou `private`

```
class Personne {  
    public string nom;  
    public int age;  
    public string metier;  
    private float taille ;  
}
```



Attributs private

De la même manière, un attribut private ne peut être utilisé que dans la classe où il est déclaré.



Attributs public

Un attribut public peut etre utiliser dans toute le programme.

Exemple :

```
Car peugot = new Car();
Console.WriteLine(peugot.wheel); // affichera 4
Console.WriteLine(peugot.serial_number); // error --> attribut private
```

```
public class Car{
public int wheel = 4;
private int serial_number = 3201244;
}
```



Encapsulation

L'encapsulation c'est le fait de donner ou non accès aux attributs à l'utilisateur.

Lorsque vous créez une classe, tous les attributs n'ont pas vocations à être accessibles par l'utilisateur.

Il est important d'organiser ses attributs privés/public pour protéger vos classes. Et garantir une meilleure expérience à l'utilisateur.



getter / setter

Les getter et setter permet de definir le retour et l'attribution d'un attribut.

`get` va retourner un attribut, et `set` va declarer/initialiser un attribut.

`get` et `set` sont des fonctions.



Utilisation get

Dans l'exemple précédent, une voiture aura toujours 4 roues, donc on pourrait définir le retour de `wheel` à 4.

```
Car peugeot = new Car();
Console.WriteLine(peugeot.wheel);

public class Car{
    public int wheel {
        get {return 4;}
    }
}
```

Un getter peut aussi retourner une variable de classe (même private vu qu'il s'agit d'une fonction au sein de la classes)

```
Car peugot = new Car();
Console.WriteLine(peugot.wheel);
Console.WriteLine(peugot.name);

public class Car{
private string car_name = "peugot";
public int wheel {
    get {return 4;}
}
public string name {
    get { return car_name;}
}
}
```



Utilisation set

Le setter permet d'attribuer une valeur à une variable de classe.

```
Car peugot = new Car();
peugot.wheel = 15; // attribution ici
Console.WriteLine(peugot.wheel);
public class Car{
private string carname = "peugot";
private int _wheel;
public int wheel {
    get {return _wheel;}
    set {_wheel = value;} // possible grâce au set
}
}
```



Methode

Une methode est une fonction qui est declaree dans une classe.

Elle peut etre `public` ou `private`. (toujours pour rendre la methode utilisable en dehors de la classe ou non)



Methode

```
Personne steve = new Personne();
Console.WriteLine(steve.Presentation());

public class Personne {
    public string nom = "Steve";
    public int age = 32;
    public string metier = "developper";
    private float taille = 1.76F; // F suffixe pour float

    public string Presentation() {
        return ("Je m'appel " + nom + ", j'ai " + age + " et mon metier est " + metier);
    }
}
```



Methode avec des Parametres

Une methode peut prendre des parametres, tout comme une fonction.

```
Personne steve = new Personne();
Console.WriteLine(steve.Presentation("Steve", 32, "developper"));

public class Personne {
    public string Presentation(string nom, int age, string metier) {
        return ("Je m'appel " + nom + ", j'ai " + age + " et mon metier est " + metier);
    }
}
```



Overload

Une methode peut etre redefinie avec des argument differents.

C'est ce qu'on appelle l'overload (surcharge en FR?) de methode.

```
Personne steve = new Personne();
Console.WriteLine(steve.Presentation("Steve", 32, "developper"));
Console.WriteLine(steve.Presentation("Steve", 32));

public class Personne {
    public string Presentation(string nom, int age, string metier) { // methode de base presentation
        return ("Je m'appel " + nom + ", j'ai " + age + " et mon metier est " + metier);
    }

    public string Presentation(string nom, int age) { // overload de la methode presentation avec 2 arguments au lieu de 3
        return ("Je m'appel " + nom + ", j'ai " + age);
    }
}
```



Constructeur

Un constructeur est une méthode qui s'exécute à l'instanciation d'une classe.

`new` permet d'instancier une classe.

Le constructeur permet d'instancier des variables de classe en même temps que la déclaration de la variable de classe.

Le constructeur est une méthode qui porte le nom de la classe

Passant de

```
Personne steve = new Personne();
Console.WriteLine(steve.nom);
public class Personne {
    public string nom = "Steve";
    public int age = 32;
    public string metier = "developper";
    private float taille = 1.76F;
}
```

```
Personne steve = new Personne();
Console.WriteLine(steve.nom);

public class Personne {
    public string nom;
    public int age;
    public string metier;
    private float taille;

    public Personne() { // constructeur
        nom = "Steve";
        age = 32;
        metier = "developper";
        taille = 1.76F;
    }
}
```



Parametres du constructeur

Le constructeur peut prendre des paramètres, comme un fonction.

Toute les personnes ne s'appelle pas Steve et n'ont pas 32 ans.

Nous allons creer un constructeur qui va `set` notre personne,
histoire de la rendre un peu plus unique

```
Personne steve = new Personne("Steve", 32, "developper", 1.73F);
Console.WriteLine(steve.nom);

public class Personne {
    public string nom;
    public int age;
    public string metier;
    private float taille;

    public Personne(string c_nom, int c_age, string c_metier, float c_taille) { // constructeur
        nom = c_nom;
        age = c_age;
        metier = c_metier;
        taille = c_taille;
    }
}
```



Parametres du constructeur

Maintenant vous pouvez créer une variable de type Personne en la nommant comme vous le souhaitez sans avoir à toucher au code dans la classe.



Overload de Constructeur

Un constructeur peut etre redefini avec des argument differents.
Comme une fonction.

C'est ce qu'on appelle l'overload de constructeur.

```
Personne steve = new Personne("Steve", 32, "developper", 1.73F);
Personne Lola = new Personne("Lola", 29);
Console.WriteLine(steve.nom);
Console.WriteLine(Lola.nom);

public class Personne {
    public string nom;
    public int age;
    public string metier;
    private float taille;

    public Personne(string c_nom, int c_age, string c_metier, float c_taille) { // constructeur
        nom = c_nom;
        age = c_age;
        metier = c_metier;
        taille = c_taille;
    }

    public Personne(string c_nom, int c_age) { // overload du constructeur
        nom = c_nom;
        age = c_age;
    }
}
```



Static



Attribut Static

Un attribut static est une variable qui appartient à la classe et non à l'objet.

Elle ne sera pas accessible via une variable; mais par le type.

```
Console.WriteLine(Personne.nbPersonne); // affichera 0
Personne Bob = new Personne("Bob");
Console.WriteLine(Personne.nbPersonne); // affichera 1
Personne Lea = new Personne("Lea");
Console.WriteLine(Personne.nbPersonne); // affichera 2
// Console.WriteLine(Bob.nbPersonne); // affichera error
// car la static n'est pas accessible via Bob
```

```
public class Personne {
    public string nom;
    public static int nbPersonne = 0;

    public Personne(string c_nom) {
        nom = c_nom;
        nbPersonne++;
    }
}
```



Heritage



Qu'est ce que l'héritage

L'héritage permet de reutiliser du code en créant des classes enfant à partir de classes parent.

La classe enfant va pouvoir utiliser les variables et les méthodes de la classe parent.

Elle peut aussi redéfinir les méthodes,

Ajouter de nouvelle variable,

La classe parent peut imposer la définition de méthode dans la classe enfant.



A quoi ça sert

Vous voulez créer un jeu vidéo de combat et pour cela vous avez besoin de :

📌 Personnages

Vous voulez avoir plusieurs classes de Personnages :

📌 Magicien

📌 Guerrier

📌 Ninja



A quoi ça sert

Chaque une d'elle aura des propriete de bases telque:

- 📌 nom
- 📌 pv (point de vie)
- 📌 agilite (% d'esquive des attaques subit)
- 📌 degat (attaque)

Maintenant vous pourriez me dire : " pourquoi ne pas faire une seul classe avec plusieur methode, on pourrait trouver un moyen ?"



A quoi ca sert

D'accord, apres la realisation de ma classe unique je veux rajouter un personnage soigner et un envouteur;

Je devrais changer des implementation dans la classe Personnage et faire attention a ne pas casser les implementations des autre personnages deja creer.

De plus il sera plus dur d'implémenter des méthodes uniques pour certains personnages.



Avantage

- 📌 Scalabilite du code
- 📌 Separation et clarte
- 📌 Pas de multiple definition des variables et methodes
- 📌 Imposer des variables et des methodes



Comment ca marche

Lorsque l'on herite d'une classe, les attributs et les methodes de la classe parent sont accessible dans la classe enfant.

Mots clefs : `:` + nom de classe parent lors de la declaration de la classe fils.



Comment ça marche

Ici Magicien (fils) herite de Personnage (parent)

De ce fait, Magicien aura tout les attributs et méthode de Personnage

```
class Magicien : Personnage {} // class Magicien herite de class Personnage
```



Constructeur parent

Si la classe parent a un constructeur, il FAUT l'appeler dans le constructeur de la classe enfant.

Mot clef : `: base`

```
class Magicien : Personnage {  
    public Magicien(string c_nom) : base(c_nom) { // va appeler le constructeur  
        // de Personnage(string c_nom)  
  
        // code  
    }  
}
```



Exemple

```
class Personnage {  
    public string nom;  
    public int pv;  
    public int agilite;  
    public int degat;  
  
    public Personnage(string c_nom) {  
        nom = c_nom;  
        pv = 100;  
        agilite = 10;  
        degat = 10;  
    }  
  
    public void presentation() {  
        Console.WriteLine("Je suis " + nom + " et j'ai " + pv + " points de vie.");  
    }  
}
```

```
class Magicien : Personnage {
public int mana;

public Magicien(string c_nom) : base(c_nom) {
    mana = 100;
}
}

class Gueurier : Personnage {
public int armure;

public Gueurier(string c_nom) : base(c_nom) {
    armure = 10;
    pv = 250;
    agilite = 5;
}
}

class Ninja : Personnage {
public int esquive;

public Ninja(string c_nom) : base(c_nom) {
    esquive = 100;
    agilite = 34;
    pv = 90;
}
}
```



Protected

Un attribut ou une methode protected est accessible dans la classe parent et dans la classe enfant.

Contrairement a un attribut ou methode private qui ne peut etre accessible dans une classes herite.

Exemple :

```
class A {  
protected int nbr_protected = 2;  
private int nbr_private = 3;  
  
public A() {  
    Console.WriteLine(nbr_protected); // affichera 2  
    Console.WriteLine(nbr_private); // affichera 3  
}  
}  
  
class B : A {  
public B() : base() {  
    Console.WriteLine(nbr_protected); // affichera 2  
    Console.WriteLine(nbr_private); // error ---> attribut private d'une autre class  
}  
}
```



Classes abstraites

Une classe abstraite est une classe qui ne peut pas etre instancier.

Elle sert de base a d'autre classe qui vont l'heriter.

Une classe abstraite peut contenir des attributs et des methodes.

Exemple : On ne veux pas de personnage qui n'appartienne a aucune classes deriver de personnage

```
abstract class Personnage {  
    public string nom;  
    public int pv;  
    public int agilite;  
    public int degat;  
  
    public Personnage(string c_nom) {  
        nom = c_nom;  
        pv = 100;  
        agilite = 10;  
        degat = 10;  
    }  
  
    public void presentation() {  
        Console.WriteLine("Je suis " + nom + " et j'ai " + pv + " points de vie.");  
    }  
  
    abstract public void attaquer();  
}
```

```
class Magicien : Personnage {
public int mana;

public Magicien(string c_nom) : base(c_nom) {
    mana = 100;
}

public override void attaquer() {
    // code
}
}

class Gueurier : Personnage {
public int armure;

public Gueurier(string c_nom) : base(c_nom) {
    armure = 10;
    pv = 250;
    agilite = 5;
}

public override void attaquer() {
    // code
}
}

class Ninja : Personnage {
public int esquive;

public Ninja(string c_nom) : base(c_nom) {
    esquive = 100;
    agilite = 34;
    pv = 90;
}

public override void attaquer() {
    // code
}
}
```



Override

Une methode peut aussi etre abstract.

Cela veut dire que la methode doit etre redefini dans la classe enfant.

Si une methode est abstract dans la classe parent et n'est pas defini dans la classes enfant, le code ne fonctionnera pas.

Mot clef : **override**



Interfaces

Une interface est une classe abstraite qui ne contient que des méthodes abstraites.

Une interface peut être implementer dans une classe.

Dans ta classe abstraite il peut y avoir des fonctions/méthodes avec une implémentation.

Tu peux également avoir des variables, alors que les interfaces ne peuvent avoir que des propriétés.

Tu ne peux hériter que d'une seule classe abstraite, alors qu'avec les



Exemple

```
interface IDéplacer {  
    public void déplacer();  
}
```

```
interface IAttaquer {  
    public void attaquer();  
}
```

```
abstract class Personnage {  
    public string nom;  
    public int pv;  
    public int agilite;  
    public int degat;  
  
    public Personnage(string c_nom) {  
        nom = c_nom;  
        pv = 100;  
        agilite = 10;  
        degat = 10;  
    }  
  
    public void presentation() {  
        Console.WriteLine("Je suis " + nom + " et j'ai " + pv + " points de vie.");  
    }  
}
```

```
class Ninja : Personnage, IDeplacer, IAttaquer {  
    public int esquive;  
  
    public Ninja(string c_nom) : base(c_nom) {  
        esquive = 100;  
        agilite = 34;  
        pv = 90;  
    }  
  
    public void attaquer() {  
        // code  
    }  
  
    public void deplacer() {  
        // code  
    }  
}
```