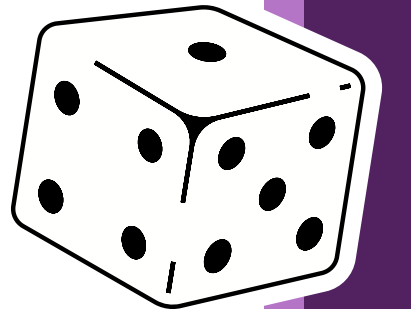


AIX YNOV CAMPUS

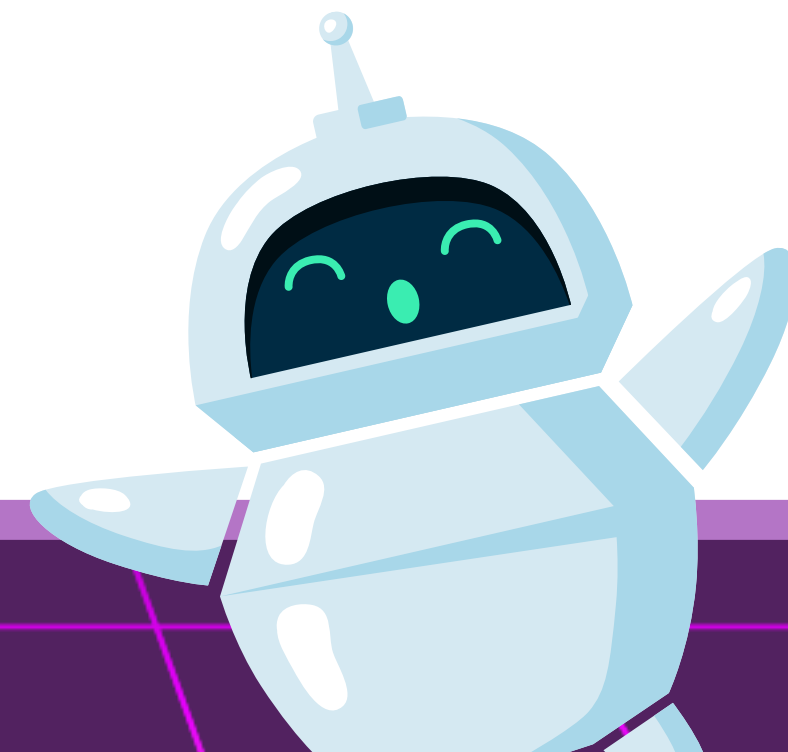
Les Bases



C SHARP

START

Par Ethan ZERAH





# HELLO WORLD

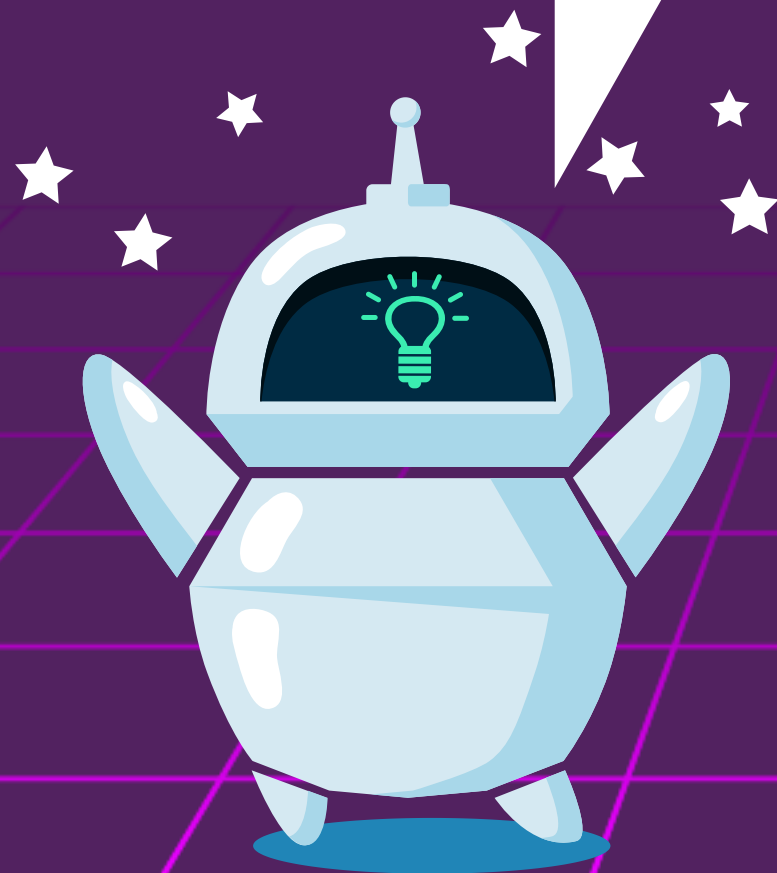
```
using System;

namespace coursCSharp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            // Affiche HelloWorld dans le terminal
            Console.WriteLine("HelloWorld");
        }
    }
}
```

```
Console.WriteLine("Hello World!");
```

READY

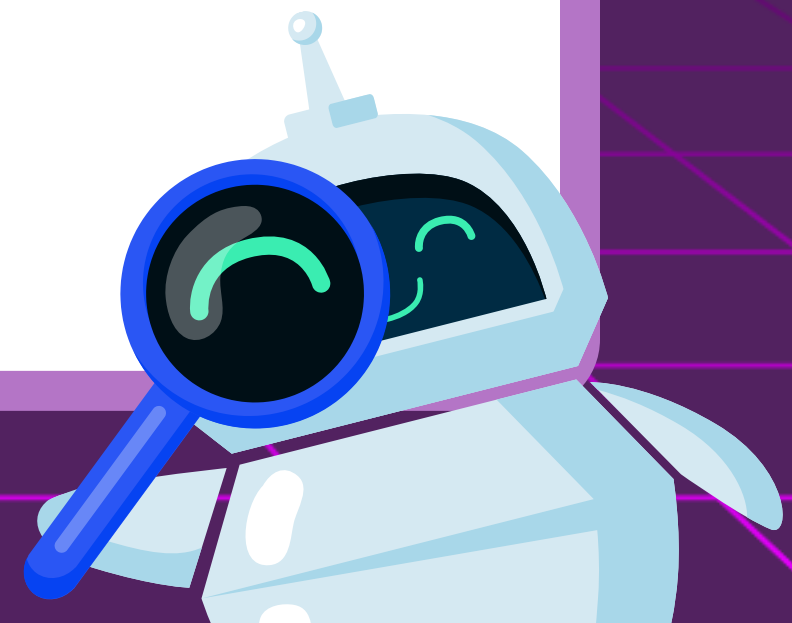
LET'S  
TRY  
THINGS!



# VARIABLES ET CONSTANTES

Pour toutes les instructions suivantes, affichez le résultat à chaque étape et décrivez ce qu'il se passe en commentaires :

- Créez une variable et donnez lui la valeur 50
- Créez une variable de valeur "Bonjour"
- Créez une variable de valeur 3.1415
- Créez une variable de valeur 'a'
- Créez une variable de valeur véridique
- Créez une constante et donnez lui la valeur 10. Donnez lui la valeur 15
- Créez deux variables x et y, donnez leur les valeurs 10 et 50. Affichez les résultats des opérations mathématiques classiques
- Concaténez deux variables textuelles de 2 façons différentes
- Initialisez 3 variables en même temps
- Récupérez le 3<sup>e</sup> caractère d'une variable textuelle
- Récupérez une entrée utilisateur dans une variable que vous avez créé



# VARIABLES

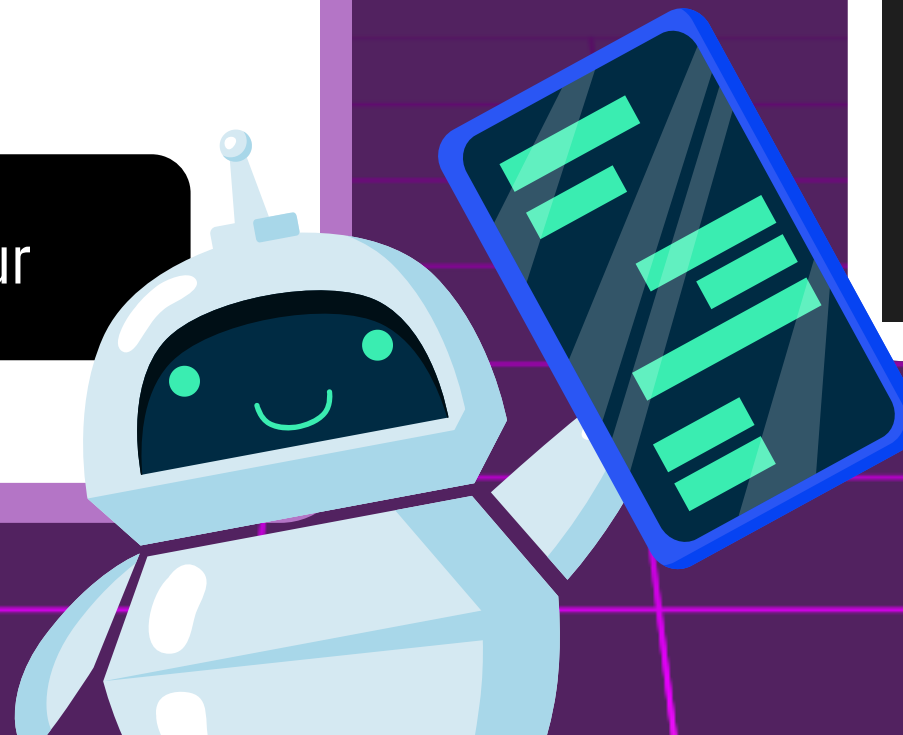
Conteneurs modifiables pour stocker les valeurs de vos données.

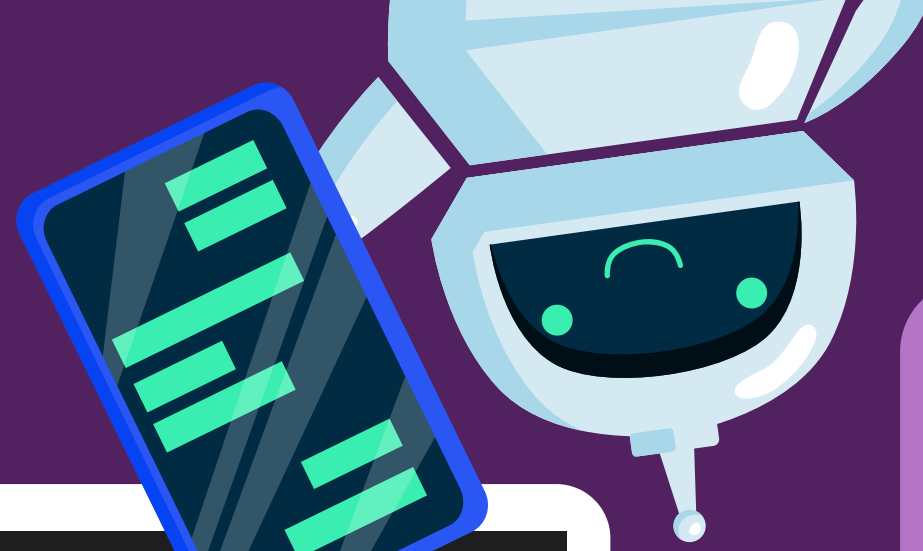
Ils sont de différents types en fonction de vos besoins.

Syntaxe :

```
type nomEnLowerCamelCase = valeur
```

```
int myInt = 10;  
long myLong = 100L;  
  
float myFloat = 1.5F;  
double myDouble = 1.6D;  
  
bool myBool = true;  
  
char myChar = 'a';  
string myString = "Hello";  
  
int x = 5, y = 6;
```





```
const int myInt = 10;  
const long myLong = 100L;  
  
const float myFloat = 1.5F;  
const double myDouble = 1.6D;  
  
const bool myBool = true;  
  
const char myChar = 'a';  
const string myString = "Hello";
```

# CONSTANTES

Conteneurs non modifiables pour stocker les valeurs de vos données.

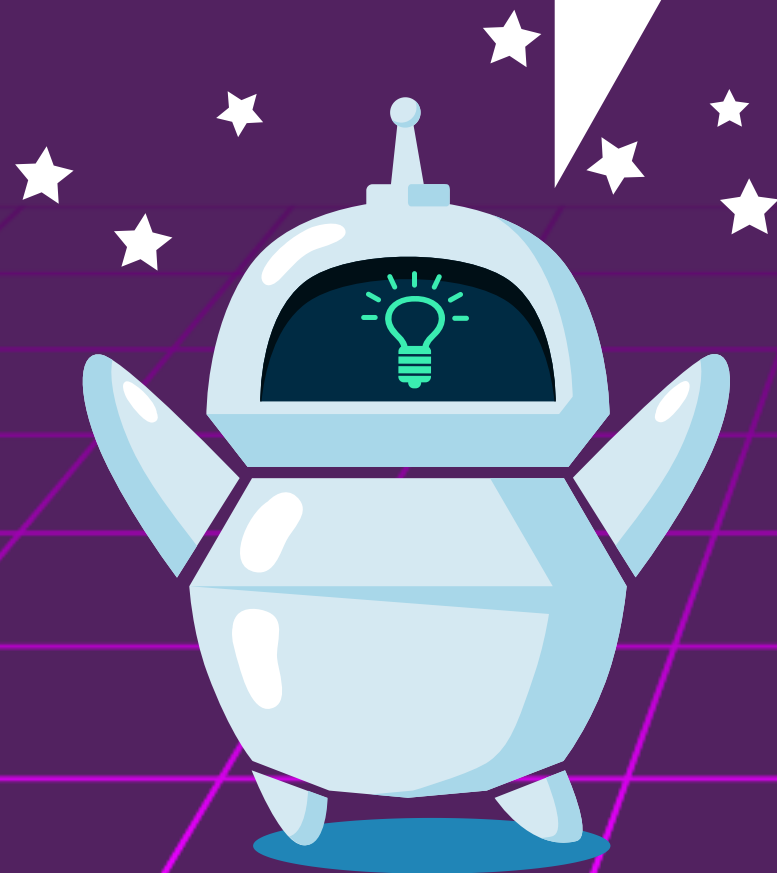
Ils sont de différents types en fonction de vos besoins.

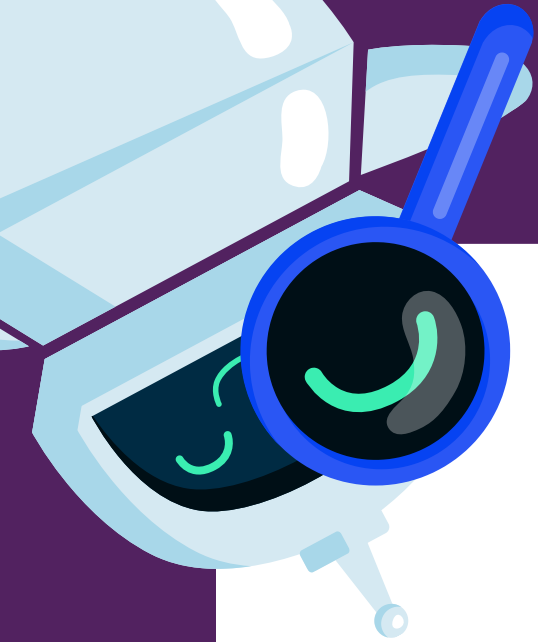
Syntaxe :

```
const type nomEnLowerCamelCase = valeur
```

READY

LET'S  
TRY  
THINGS!





# LES CONDITIONS

Pour toutes les instructions suivantes, affichez le résultat à chaque étape et décrivez ce qu'il se passe en commentaires :

- Affichez "HelloWorld" si x est supérieur à y
- Affichez "1" si x est supérieur à y sinon affichez "2"
- Affichez "1" si x est supérieur à y sinon si x est supérieur à y affichez "2" sinon affichez "3"
- Initialisez un bool grâce à une condition de 2 manières différentes
- Affichez des messages différents en fonction de la valeur d'un int (1 à 5)
- Ajoutez un message par default pour les valeurs différentes



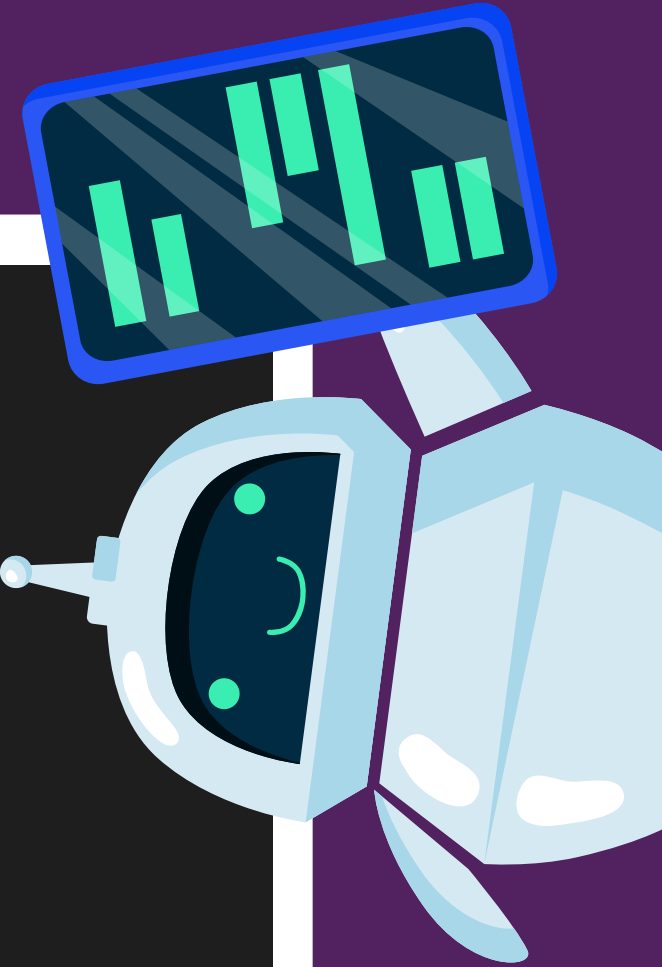
# IF... ELSE

Condition basique permettant de prévoir plusieurs possibilités

Syntaxes :

```
if (Condition)  
{  
    Action  
}
```

*Condition ? Valeur Vraie : Valeur Fausse;*



```
int number = 5;  
bool isNull;  
if (number == 0)  
{  
    isNull = true;  
}  
else  
{  
    isNull = false;  
}  
  
isNull = (number == 0) ? true : false ;  
  
int x = 5,y =6;  
  
string resultA = x==y ? "Vrai" : "Faux";  
  
string resultB = x==y ? "VraiA" : x>y ? "VraiB" : "Faux";
```



```
int place = 2;
switch (place)
{
    default:
        Console.WriteLine("Somewhere");
        break;
    case 1:
        Console.WriteLine("Aix");
        break;
    case 2:
        Console.WriteLine("NewYork");
        break;
}
```

# SWITCH

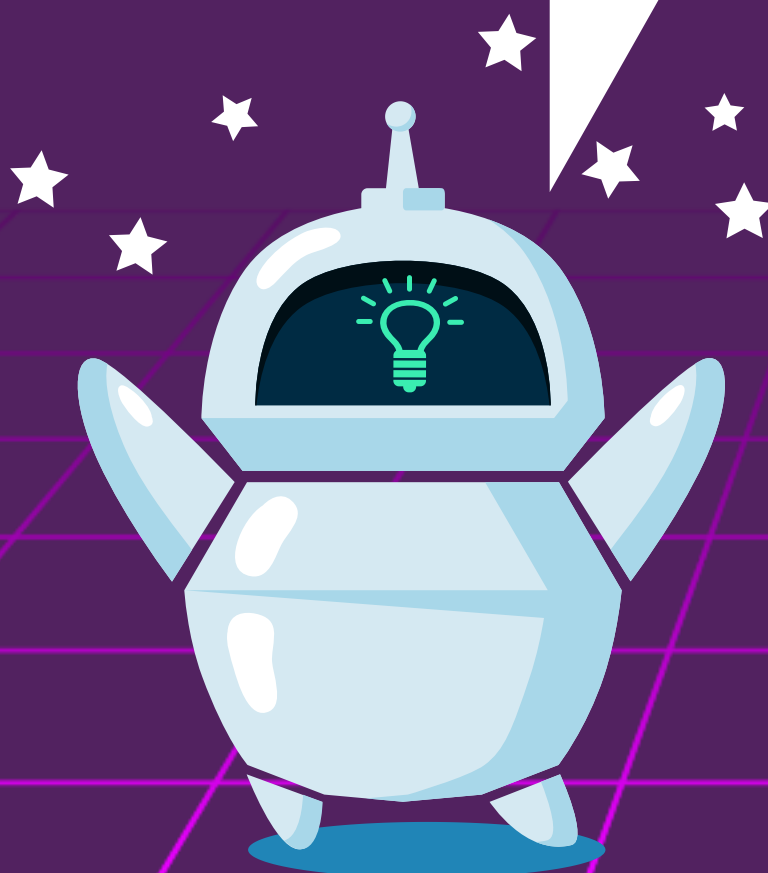
Condition permettant de lister tous les cas possibles

Syntaxe :

```
switch (var)
{
    case Valeur:
        Action;
        Break;
    default:
        Action;
        Break;
}
```

READY

LET'S  
TRY  
THINGS!





# LES BOUCLES

Pour toutes les instructions suivantes, affichez le résultat à chaque étape et décrivez ce qu'il se passe en commentaires :

- Créez une boucle qui affichera 5 fois un message
- Créez une boucle conditionnelle
- Faites en sorte que votre boucle conditionnelle aie une première itération avant de rentrer dans la boucle
- Récupérez un entier en entrée utilisateur qui sera passé en condition d'une boucle avec compteur. Si le nombre est égal au compteur alors sortez de la boucle, si le nombre est égal au compteur moins 3 alors passez à la prochaine itération, affichez le compteur.
- Créez une boucle conditionnelle dans une boucle avec compteur et initialisez des valeurs modifiées dans chacune des boucles

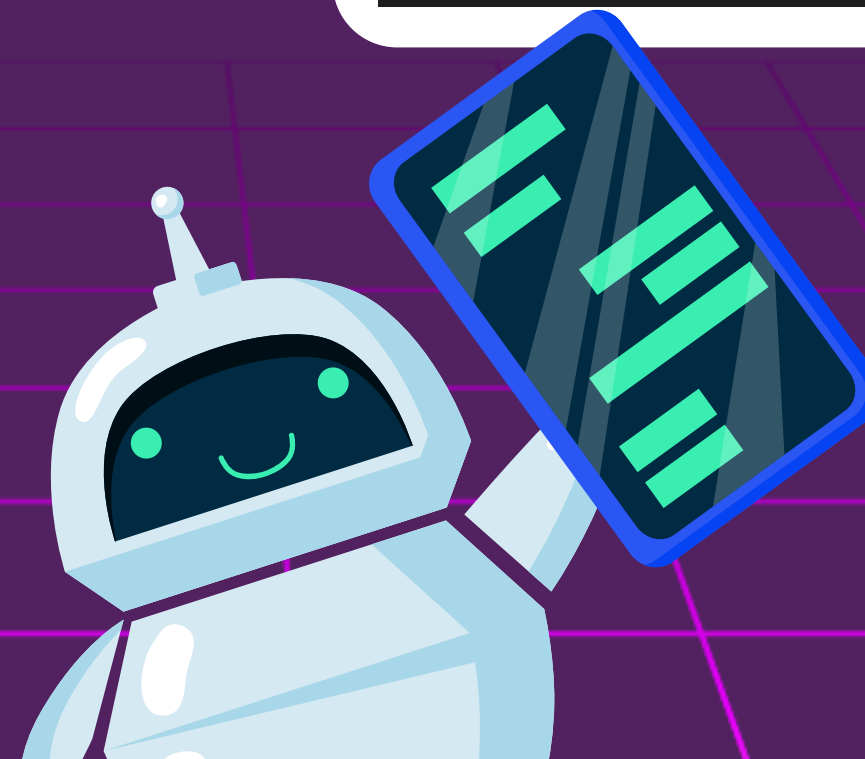
# FOR

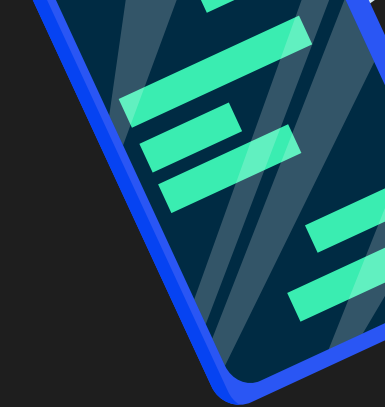

Boucle permettant d'effectuer des actions un nombre de fois données.

Syntaxe :

```
for (Initialisation; Condition; Incrémentation)  
{  
    Action  
}
```

```
for(int i = 0; i < 5; i++)  
{  
    Console.WriteLine(i);  
}
```





```
int i = 1;
while (i < 6)
{
    Console.WriteLine(i);
    i++;
}
int i = 1;
do
{
    Console.WriteLine(i);
    i++;
}
while(i < 6);
```

# WHILE

Boucle permettant d'effectuer des actions un nombre de fois indéfini.

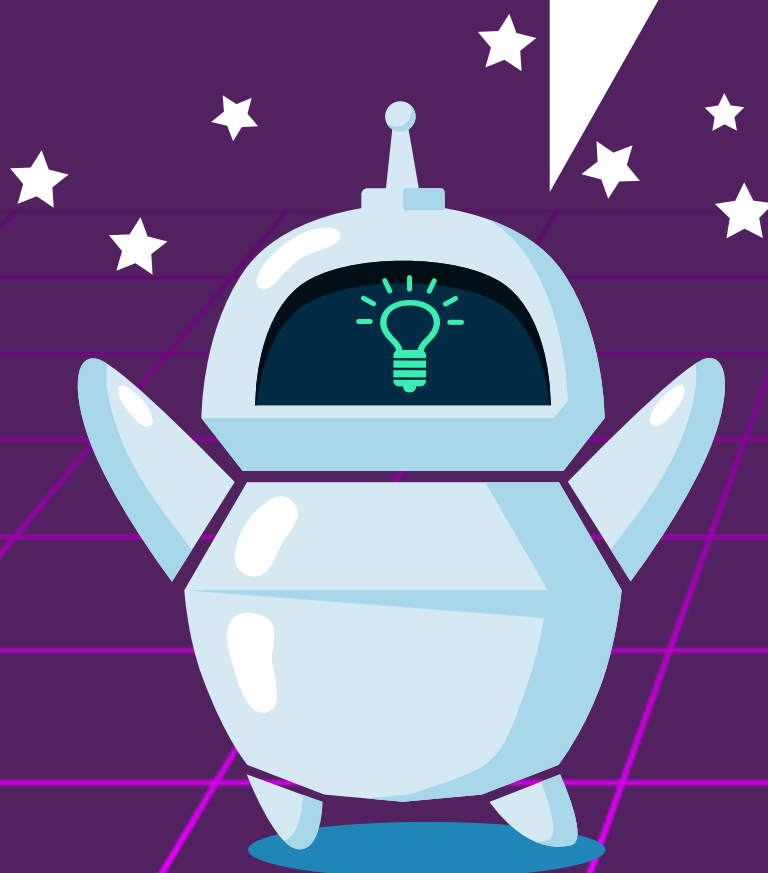
Syntaxe :

```
while (Condition)
{
    Action
}
```

```
do
{
    Action
}
while (Condition)
```

READY

LET'S  
TRY  
THINGS!





# LES TABLEAUX



Pour toutes les instructions suivantes, affichez le résultat à chaque étape et décrivez ce qu'il se passe en commentaires :

- Créez un tableau comprenant 6 fruits
- Créez un tableau comprenant 5 char de 2 manières différentes
- Créez un tableau comprenant 5 nombres désordonnés puis organisez les dans l'ordre croissant
- Créez un tableau à 2 dimensions
- Créez un mini programme demandant à l'utilisateur de rentrer des valeurs jusqu'à ce qu'il décide de s'arrêter, il les rentre dans un tableau et puis affiche le tableau en inversé
- Créez un tableau comprenant 3 nombres générés aléatoirement (1 à 6), si deux nombres ou plus sont égaux alors ceux ci sont retirés du tableau. S'il ne reste plus qu'un seul nombre, il faut qu'il soit égal à 6 pour être retiré. A chaque étape, les nombres sont régénérés et affichés avec un compteur représentant le nombre d'essais. Le programme se termine lorsqu'il n'y a plus de nombre dans le tableau



# TABLEAUX

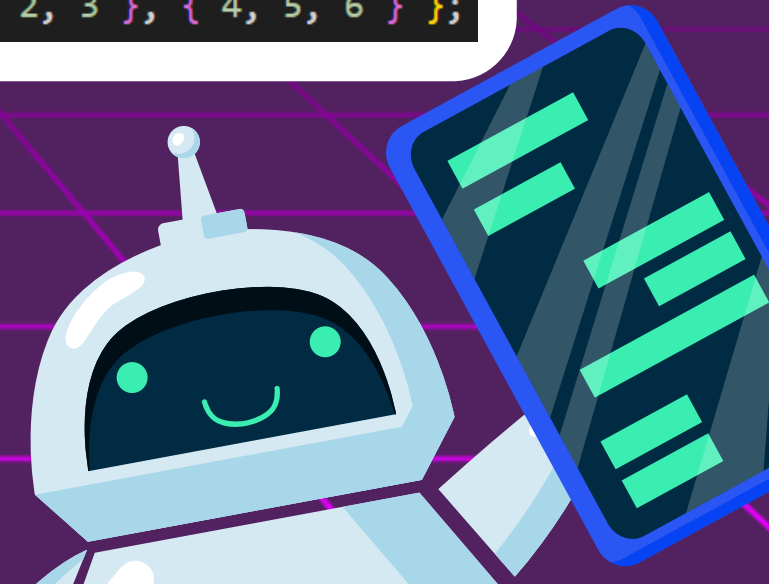
Type de variable permettant de stocker diverses valeurs d'un même type qui seront accessibles grâce à leur index.

Syntaxes :

```
type[] nameVar = {Valeur1, Valeur2, ...}
```

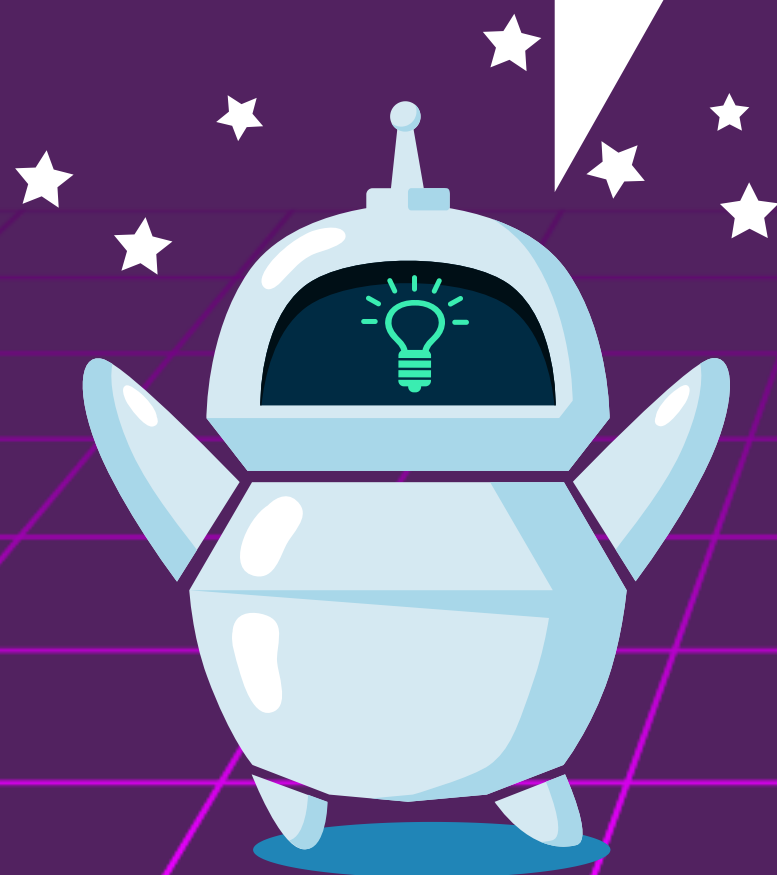
```
type[] nameVar = new type[IndexMax]
```

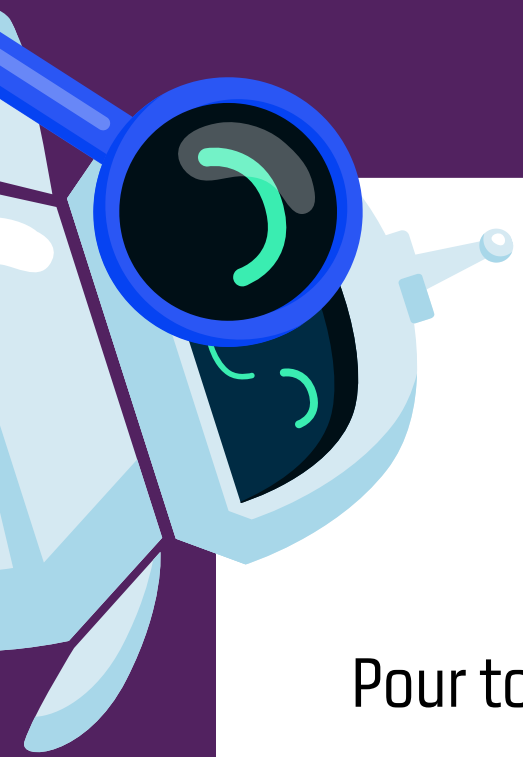
```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
int[] nums = {10, 20, 30, 40};
string[] fruits;
fruits = new string[5]{"", "", "", "", ""};
int[][] tableauEscalier =
{
    new int[] { 1, 3, 5, 7, 9 },
    new int[] { 0, 2, 4, 6 },
    new int[] { 11, 22 }
};
int[,] multiDimensionalArray = new int[2, 3]{ { 1, 2, 3 }, { 4, 5, 6 } };
```



READY

LET'S  
TRY  
THINGS!





# LES MÉTHODES

Pour toutes les instructions suivantes, affichez le résultat à chaque étape et décrivez ce qu'il se passe en commentaires :


- Créez une méthode permettant d'afficher "HelloWorld" lorsqu'elle est appelée
- Créez une méthode permettant d'additionner 3 nombres passés en paramètres et de renvoyer le résultat
- Créez deux méthodes récursives. Faites attention à laisser une condition de sortie pour ne pas créer une boucle infinie
- Créez deux méthodes de même nom permettant respectivement d'ajouter deux int et deux double passés en paramètres et affichera le résultat
- Créez deux méthodes effectuant des actions puis renvoient une variable et appelez les depuis le main
- Créez une méthode permettant d'additionner toutes les valeurs d'un tableau et de renvoyer le résultat
- Créez une méthode calculant la suite de Fibonacci jusqu'à la valeur donnée par l'utilisateur, il faudra tester la valeur rentrée par l'utilisateur pour ne pas avoir d'erreur
- Refaites l'exercice d'avant en utilisant des méthodes récursives

# MÉTHODES

Bloc de code qui ne tourne que quand il est appelé. Il est possible de passer des paramètres et de faire retourner une valeur à n'importe quel moment.

Syntaxe :

```
static ReturnType MethodName(type paramName)
{
    Action;
}
```



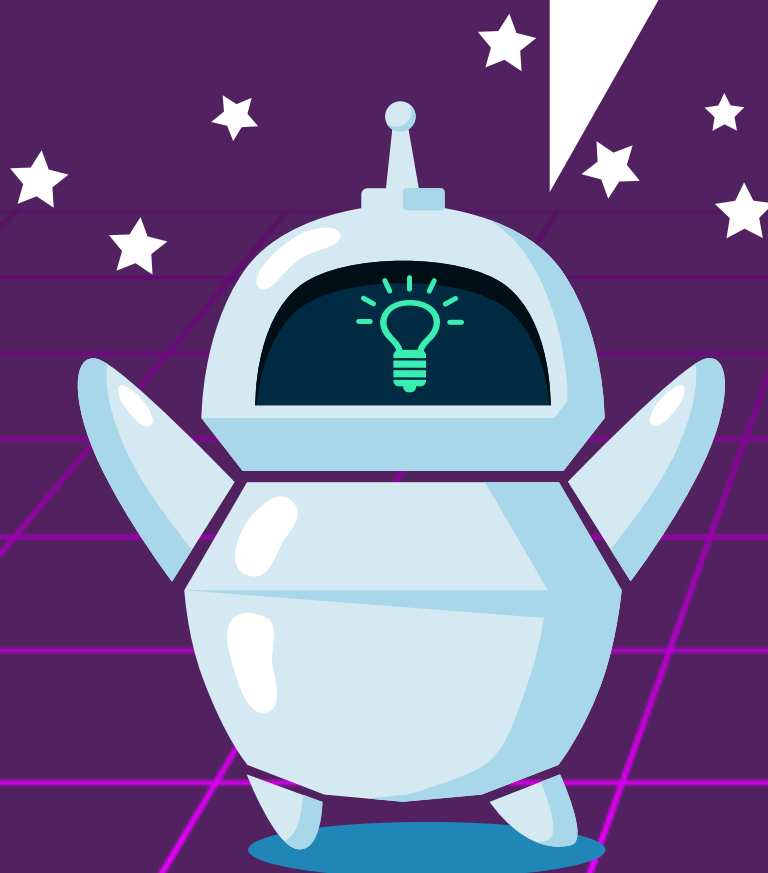
```
static void HelloWorld()
{
    Console.WriteLine("HelloWorld");
}

static int Add(int x, int y)
{
    return x+y;
}

int sum = Add(50,10);
```

READY

LET'S  
TRY  
THINGS!

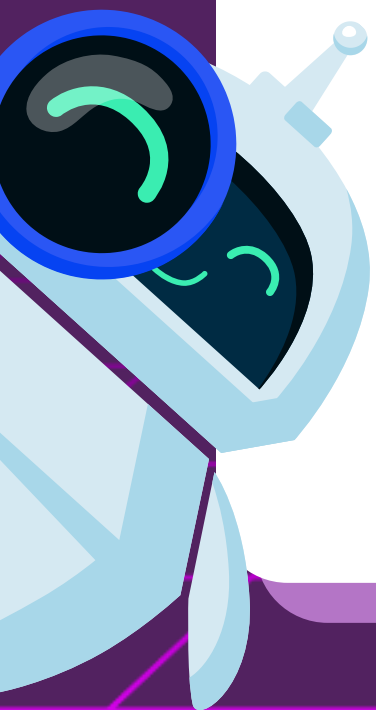




# LES CLASSES ET OBJETS

Pour toutes les instructions suivantes, affichez le résultat à chaque étape et décrivez ce qu'il se passe en commentaires :

- Créez une classe Voiture puis un objet de cette classe
- Ajoutez une variable couleur et une variable année. Initialisez les pour votre objet
- Créez un objet tout en initialisant toutes ses variables
- Ajoutez une méthode permettant d'afficher toutes les variables de votre objet. Appelez la
- Ajoutez une variable accessible uniquement depuis cette classe
- Créez une propriété permettant d'accéder et de modifier une variable inaccessible
- Créez une classe fruit (nom) héritant de la classe aliment (origine, manger())
- Créez deux classes héritant d'une autre et faites en sorte d'avoir une même méthode qui affiche différentes choses en fonction de la classe appelante
- Créez une classe avec une méthode abstraite ainsi qu'une interface
- Créez une énumération



# CLASSES ET OBJETS

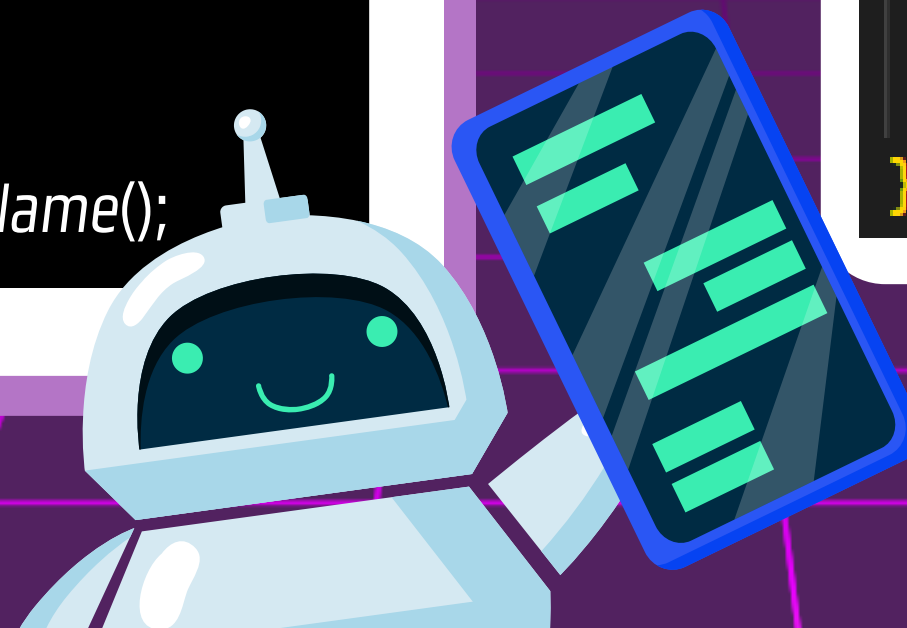
Une classe correspond à un modèle à suivre avec ses champs et méthodes comme les types de variables. Un objet suit ce modèle.

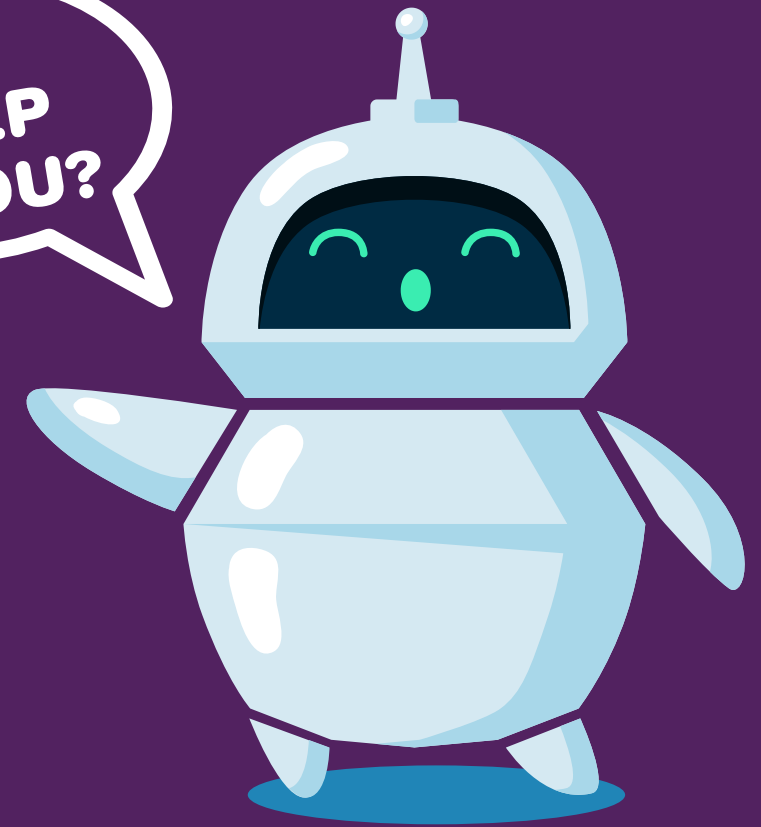
Syntaxe :

```
class ClassName
{
    Champs;
    Méthodes();
}
```

```
ClassName ObjectName = new ClassName();
```

```
class Car
{
    1 reference
    public string color = "red";
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Car myObj = new Car();
        Console.WriteLine(myObj.color);
    }
}
```





MERCI DE  
VOTRE ÉCOUTE!

