Ce qu'il faut savoir sur les conditions.

Concernant les opérations de comparaisons

Je peux comparer des données en utilisant les opérateurs de comparaisons.

👇

> plus grand que

>= plus grand ou égal que

< plus petit que

<= plus petit ou égal que

== égal à

!= différent de

Concernant les `bool` ou `booléens`.

Si je fais le code suivant 👇 :

var foo = 5;var bar = 15;var isHigher = foo >= bar;

Je cherche à savoir si ma variable `foo` est supérieur ou égale à ma variable bar. Vraisemblablement `5` n'est pas supérieur à `15`. Par conséquent, le résultat de mon opération de comparaison vaudra `false` (faux) et sera stocké dans ma variable `isHigher` qui sera donc du type `bool` pouvant valoir `true` ou `false`.

C'est ce type de donnée qui me permettra de travailler avec des conditions.

Concernant les conditions

Si je veux créer un bout de code qui s'exécute **seulement si une condition est rempli**e, je peux le faire en écrivant :

if(*ma condition est vraie*){ //monboutdecode}

Et je remplacerai bien évidemment *ma condition est vraie* par un `bool` ou opération de comparaisons.

Si je veux qu'un bout de code s'exécute si une condition est remplie et si je veux qu'un autre bout de code s'exécute si elle n'est pas remplie je peux faire :

```
if(*ma condition est vraie*){
 // monboutdecode
} else {
 // monautreboutdecode
}
```

Je n'ai pas de limites dans le nombre de cas que je peux écrire.

Par exemple

```
if(*ma condition est vraie*){
// monboutdecode
} else if (*ma deuxième condition est vraie*) {
 // monautreboutdecode
} else if (*ma troisième condition est vraie*) {
 // encoreunautreboutdecode
}
else {
 // mondernierboutdecode
}
```

Me permet de détailler **plusieurs cas possibles** les uns à la suite de l'autre.

La différence entre une suite de `if` et un `if / else` c'est que dans mon deuxième cas (`if / else`), c'est **la première condition qui est vraie qui sera exécutée**, alors que dans mon premier cas(`if`), il peut en théorie en avoir plusieurs de vraies.

A la place du code précédent, je peux également utiliser un switch qui permet de rendre le code un peu plus parlant surtout si le nombre de conditions devient important. Je peux réécrire ce code de la manière suivante :

```
switch(macondition){
case 'maconditionvautX':
 // monboutdecode
break;
  case 'maconditionvautY':
 // monautreboutdecode
 break;
case 'maconditionvautZ':
 // encoreunautreboutdecode
 break;
 default:
 // mondernierboutdecode
break;}
```

Concernant la commande Console.Readkey()

J'ai mentionné dans une de mes vidéos précédentes qu'à la place de `Console.ReadLine()` qui permet de récupérer une saisie composé e de plusieurs caractères, j'aurai pu utiliser `Console.Readkey()` qui permet de lire directement la touche saisie par l'utilisateur. **C'est vrai. Mais son utilisation demande quelques notions que nous aborderons plus loin dans la formation.** Patience ! ⏳

Une autre syntaxe : Les opérations ternaires

Si on considère le code suivant :

```
int age = 20;

string message = "";

if(age < 18) {
  message = "Vous êtes mineur(e) !";
} else {
  message = "Vous êtes majeur(e) !";
}
```

On pourrait, l'écrire d'une manière différente, plus concise.

```
int age = 20;

bool message = age < 18 ? "Vous êtes mineur(e) !" : "Vous êtes majeur(e) !";
```

L'élément important à comprendre ici est que la condition `if else` **peut s'écrire d'une autre manière sous la forme suivante :**

```
condition ? si_vraie : si_faux;
```

On appelle **? l'opérateur ternaire** dans ce cas. Pratique lorsqu'on souhaite écrire un code plus concis !