Apprenez à programmer en C !

40 heures Moyenne

Mis à jour le 29/07/2019

Les conditions

Nous avons vu dans le premier chapitre qu'il existait de nombreux langages de programmation. Certains se ressemblent d'ailleurs : un grand nombre d'entre eux sont inspirés du langage C.

En fait le langage C a été créé il y a assez longtemps, ce qui fait qu'il a servi de modèle à de nombreux autres plus récents.
La plupart des langages de programmation ont finalement des ressemblances, ils reprennent les principes de base de leurs aînés.

En parlant de principes de base : nous sommes en plein dedans. Nous avons vu comment créer des variables, faire des calculs avec (concept commun à tous les langages de programmation !), nous allons maintenant nous intéresser aux **conditions**.
Sans conditions, nos programmes informatiques feraient toujours la même chose !

La condition if... else

Les conditions permettent de tester des variables. On peut par exemple dire « si la variable machin est égale à 50, fais ceci »... Mais ce serait dommage de ne pouvoir tester que l'égalité ! Il faudrait aussi pouvoir tester si la variable est inférieure à 50, inférieure ou égale à 50, supérieure, supérieure ou égale... Ne vous inquiétez pas, le C a tout prévu !

Pour étudier les conditions `if... else`, nous allons suivre le plan suivant :

1. quelques symboles à connaître avant de commencer,
2. le test `if`,

3. le test `else`,
4. le test `else if`,
5. plusieurs conditions à la fois,
6. quelques erreurs courantes à éviter.

Avant de voir comment on écrit une condition de type `if... else` en C, il faut donc que vous connaissiez deux ou trois symboles de base. Ces symboles sont indispensables pour réaliser des conditions.

Quelques symboles à connaître

Voici un petit tableau de symboles du langage C **à connaître par coeur** :

Symbole	Signification
==	est égal à
>	est supérieur à
<	est inférieur à
>=	est supérieur ou égal à
<=	est inférieur ou égal à
!=	est différent de

> Faites très attention, il y a bien deux symboles `==` pour tester l'égalité. Une erreur courante que font les débutants est de ne mettre qu'un symbole `=`, ce qui n'a pas la même signification en C. Je vous en reparlerai un peu plus bas.

Un `if` simple

Attaquons maintenant sans plus tarder. Nous allons faire un test simple, qui va dire à l'ordinateur :

Citation

> SI la variable vaut ça,
> ALORS fais ceci.

En anglais, le mot « si » se traduit par `if`. C'est celui qu'on utilise en langage C pour introduire une condition.
Écrivez donc un `if`. Ouvrez ensuite des parenthèses : à l'intérieur de ces parenthèses vous devrez écrire votre condition.

Ensuite, ouvrez une accolade `{` et fermez-la un peu plus loin `}`. Tout ce qui se trouve à l'intérieur des accolades sera exécuté uniquement si la condition est vérifiée.

Cela nous donne donc à écrire :

c
```c
if (/* Votre condition */)
{
  // Instructions à exécuter si la condition est vraie
}
```

À la place de mon commentaire « Votre condition », on va écrire une condition pour tester une variable.
Par exemple, on pourrait tester une variable `age` qui contient votre âge. Tenez pour s'entraîner, on va tester si vous êtes majeur, c'est-à-dire **si votre âge est supérieur ou égal à 18** :

c
```c
if (age >= 18)
{
  printf ("Vous etes majeur !");
}
```

Le symbole `>=` signifie « supérieur ou égal », comme on l'a vu dans le tableau tout à l'heure.

> S'il n'y a qu'une instruction entre les accolades (comme c'est le cas ici), alors celles-ci deviennent facultatives. Je recommande néanmoins de toujours mettre des accolades pour des raisons de clarté.

Tester ce code

Si vous voulez tester les codes précédents pour voir comment le `if` fonctionne, il faudra placer le `if` à l'intérieur d'une fonction `main` et ne pas oublier de déclarer une variable `age` à laquelle on donnera la valeur de notre choix.

Cela peut paraître évident pour certains, mais plusieurs lecteurs visiblement perdus m'ont encouragé à ajouter cette explication. Voici donc un code complet que vous pouvez tester :

c
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int age = 20;

    if (age >= 18)
    {
      printf ("Vous etes majeur !\n");
    }

```

```
13      return 0;
14  }
```

Ici, la variable `age` vaut 20, donc le « Vous êtes majeur ! » s'affichera.
Essayez de changer la valeur initiale de la variable pour voir. Mettez par exemple 15 : la condition sera fausse, et donc « Vous êtes majeur ! » ne s'affichera pas cette fois.

Utilisez ce code de base pour tester les prochains exemples du chapitre.

Une question de propreté

La façon dont vous ouvrez les accolades n'est pas importante, votre programme fonctionnera aussi bien si vous écrivez tout sur une même ligne. Par exemple :

c

```c
if (age >= 18) {  printf ("Vous etes majeur !"); }
```

Pourtant, même s'il est possible d'écrire comme ça, c'est **absolument déconseillé**.
En effet, tout écrire sur une même ligne rend votre code difficilement lisible. Si vous ne prenez pas dès maintenant l'habitude d'aérer votre code, plus tard quand vous écrirez de plus gros programmes vous ne vous y retrouverez plus !

Essayez donc de présenter votre code source de la même façon que moi : une accolade sur une ligne, puis vos instructions (précédées d'une tabulation pour les « décaler vers la droite »), puis l'accolade de fermeture sur une ligne.

> Il existe plusieurs bonnes façons de présenter son code source. Ça ne change rien au fonctionnement du programme final, mais c'est une question de « style informatique » si vous voulez. Si vous voyez le code de quelqu'un d'autre présenté un peu différemment, c'est qu'il code avec un style différent. Le principal dans tous les cas étant que le code reste aéré et lisible.

Le `else` pour dire « sinon »

Maintenant que nous savons faire un test simple, allons un peu plus loin : si le test n'a pas marché (il est faux), on va dire à l'ordinateur d'exécuter d'autres instructions.

En français, nous allons donc écrire quelque chose qui ressemble à cela :

Citation

> SI la variable vaut ça,
> ALORS fais ceci,
> SINON fais cela.

Il suffit de rajouter le mot `else` après l'accolade fermante du `if` .
Petit exemple :

c

```c
if (age >= 18) // Si l'âge est supérieur ou égal à 18
{
  printf ("Vous etes majeur !");
}
else // Sinon...
{
  printf ("Ah c'est bete, vous etes mineur !");
}
```

Les choses sont assez simples : si la variable `age` est supérieure ou égale à 18, on affiche le message « Vous êtes majeur ! », sinon on affiche « Vous êtes mineur ».

Le `else if` pour dire « sinon si »

On a vu comment faire un « si » et un « sinon ». Il est possible aussi de faire un « sinon si » pour faire un autre test si le premier test n'a pas marché. Le « sinon si » se place entre le `if` et le `else` .

On dit dans ce cas à l'ordinateur :

Citation

> SI la variable vaut ça ALORS fais ceci,
> SINON SI la variable vaut ça ALORS fais ça,
> SINON fais cela.

Traduction en langage C :

c

```c
if (age >= 18) // Si l'âge est supérieur ou égal à 18
{
  printf ("Vous etes majeur !");
}
else if ( age > 4 ) // Sinon, si l'âge est au moins supérieur à 4
{
  printf ("Bon t'es pas trop jeune quand meme...");
}
else // Sinon...
{
  printf ("Aga gaa aga gaaa"); // Langage bébé, vous pouvez pas comprendre
}
```

L'ordinateur fait les tests dans l'ordre.

1. D'abord il teste le premier `if` : si la condition est vraie, alors il exécute ce qui se trouve entre les premières accolades.
2. Sinon, il va au « sinon si » et fait à nouveau un test : si ce test est vrai, alors il exécute les instructions correspondantes entre accolades.
3. Enfin, si aucun des tests précédents n'a marché, il exécute les instructions du « sinon ».

> Le `else` et le `else if` ne sont pas obligatoires. Pour faire une condition, seul un `if` est nécessaire (logique me direz-vous, sinon il n'y a pas de condition !).

Notez qu'on peut mettre autant de `else if` que l'on veut. On peut donc écrire :

Citation

> SI la variable vaut ça,
> ALORS fais ceci,
> SINON SI la variable vaut ça ALORS fais ça,
> SINON SI la variable vaut ça ALORS fais ça,
> SINON SI la variable vaut ça ALORS fais ça,
> SINON fais cela.

Plusieurs conditions à la fois

Il peut aussi être utile de faire plusieurs tests à la fois dans votre `if`. Par exemple, vous voudriez tester si l'âge est supérieur à 18 ET si l'âge est inférieur à 25.
Pour faire cela, il va falloir utiliser de nouveaux symboles :

Symbole	Signification
`&&`	ET
`||`	OU
`!`	NON

Test ET

Si on veut faire le test que j'ai mentionné plus haut, il faudra écrire :

c

```c
if (age > 18 && age < 25)
```

Les deux symboles `&&` signifient ET. Notre condition se dirait en français : « si l'âge est supérieur à 18 ET si l'âge est inférieur à 25 ».

Test OU

Pour faire un OU, on utilise les deux signes `||`. Je dois avouer que ce signe n'est pas facilement accessible sur nos claviers. Pour le taper sur un clavier AZERTY français, il faudra faire `Alt Gr + 6`. Sur un clavier belge, il faudra faire `Alt Gr + &`.

Imaginons pour l'exemple un programme stupide qui décide si une personne a le droit d'ouvrir un compte en banque. C'est bien connu, pour ouvrir un compte en banque il vaut mieux ne pas être trop jeune (on va dire arbitrairement qu'il faut avoir au moins 30 ans) ou bien avoir beaucoup d'argent (parce que là, même à 10 ans on vous acceptera à bras ouverts !). Notre test pour savoir si le client a le droit d'ouvrir un compte en banque pourrait être :

c

```c
if (age > 30 || argent > 100000)
{
    printf("Bienvenue chez PicsouBanque !");
}
else
{
    printf("Hors de ma vue, miserable !");
}
```

Ce test n'est valide que si la personne a plus de 30 ans ou si elle possède plus de 100 000 euros !

Test NON

Le dernier symbole qu'il nous reste à tester est le point d'exclamation. En informatique, le point d'exclamation signifie « non ».
Vous devez mettre ce signe avant votre condition pour dire « si cela n'est pas vrai » :

c

```c
if (!(age < 18))
```

Cela pourrait se traduire par « si la personne n'est pas mineure ». Si on avait enlevé le `!` devant, cela aurait signifié l'inverse : « si la personne est mineure ».

Quelques erreurs courantes de débutant

N'oubliez pas les deux signes `==`

Si on veut tester si la personne a tout juste 18 ans, il faudra écrire :

c

```c
if (age == 18)
{
    printf ("Vous venez de devenir majeur !");
}
```

N'oubliez pas de mettre deux signes « égal » dans un `if`, comme ceci : `==`

Si vous ne mettez qu'un seul signe `=`, alors votre variable **prendra** la valeur 18 (comme on l'a appris dans le chapitre sur les variables). Nous ce qu'on veut faire ici, c'est tester la valeur de la variable, non pas la changer ! Faites très attention à cela, beaucoup d'entre vous n'en mettent qu'un quand ils débutent et forcément... leur programme ne fonctionne pas comme ils voudraient !

Le point-virgule de trop

Une autre erreur courante de débutant : vous mettez parfois un point-virgule à la fin de la ligne d'un `if`. Or, un `if` est une condition, et on ne met de point-virgule qu'à la fin d'une instruction et non d'une condition. Le code suivant ne marchera pas comme prévu car il y a un point-virgule à la fin du `if` :

c

```c
if (age == 18); // Notez le point-virgule ici qui ne devrait PAS être là
{
    printf ("Tu es tout juste majeur");
}
```

Les booléens, le coeur des conditions

Nous allons maintenant entrer plus en détails dans le fonctionnement d'une condition de type `if... else`.
En effet, les conditions font intervenir quelque chose qu'on appelle **les booléens** en informatique.

Quelques petits tests pour bien comprendre

Nous allons commencer par faire quelques petites expériences avant d'introduire cette nouvelle notion. Voici un code source très simple que je vous propose de tester :

c

```c
if (1)
{
    printf("C'est vrai");
}
else
{
    printf("C'est faux");
}
```

Résultat :

```
C'est vrai
```

> Mais ? On n'a pas mis de condition dans le `if`, juste un nombre. Qu'est-ce que ça veut dire ? Ça n'a pas de sens.

Si, ça en a, vous allez comprendre. Faites un autre test en remplaçant 1 par 0 :

c

```c
if (0)
{
    printf("C'est vrai");
}
else
{
    printf("C'est faux");
```

```
8 }
```

Résultat :

```
C'est faux
```

Faites maintenant d'autres tests en remplaçant le 0 par n'importe quel autre nombre entier, comme 4, 15, 226, -10, -36, etc.
Qu'est-ce qu'on vous répond à chaque fois ? On vous répond : « C'est vrai ».

Résumé de nos tests : si on met un 0, le test est considéré comme faux, et si on met un 1 ou n'importe quel autre nombre, le test est vrai.

Des explications s'imposent

En fait, à chaque fois que vous faites un test dans un `if`, ce test renvoie la valeur 1 s'il est vrai, et 0 s'il est faux.

Par exemple :

c

```c
1 if (age >= 18)
```

Ici, le test que vous faites est `age >= 18`.

Supposons que `age` vaille 23. Alors le test est vrai, et l'ordinateur « remplace » en quelque sorte `age >= 18` par 1.
Ensuite, l'ordinateur obtient (dans sa tête) un `if (1)`. Quand le nombre est 1, comme on l'a vu, l'ordinateur dit que la condition est vraie, donc il affiche « C'est vrai » !

De même, si la condition est fausse, il remplace `age >= 18` par le nombre 0, et du coup la condition est fausse : l'ordinateur va lire les instructions du `else`.

Un test avec une variable

Testez maintenant un autre truc : envoyez le résultat de votre condition dans une variable, comme si c'était une opération (car pour l'ordinateur, **c'est** une opération !).

c

```c
1 int age = 20;
2 int majeur = 0;
3
4 majeur = age >= 18;
5 printf("Majeur vaut : %d\n", majeur);
```

Comme vous le voyez, la condition `age >= 18` a renvoyé le nombre 1 car elle est vraie. Du coup, notre variable `majeur` vaut 1, on vérifie d'ailleurs cela grâce à un `printf` qui montre bien qu'elle a changé de valeur.

Faites le même test en mettant `age == 10` par exemple. Cette fois, `majeur` vaudra 0.

Cette variable `majeur` est un booléen

Retenez bien ceci : on dit qu'une variable à laquelle on fait prendre les valeurs 0 et 1 est **un booléen**.

Et aussi ceci :

- 0 = faux
- 1 = vrai

Pour être tout à fait exact, 0 = faux et tous les autres nombres valent vrai (on a eu l'occasion de le tester plus tôt). Ceci dit, pour simplifier les choses on va se contenter de n'utiliser que les nombres 0 et 1, pour dire si « quelque chose est faux ou vrai ».

En langage C, il n'existe pas de type de variable « booléen ».
Du coup, on est obligé d'utiliser un type entier comme `int` pour gérer les booléens.

Les booléens dans les conditions

Souvent, on fera un test `if` sur une variable booléenne :

c

```c
int majeur = 1;

if (majeur)
{
    printf("Tu es majeur !");
}
else
{
    printf("Tu es mineur");
}
```

Comme majeur vaut 1, la condition est vraie, donc on affiche « Tu es majeur ! ».

Ce qui est très pratique, c'est que la condition peut être lue facilement par un être humain. On voit `if (majeur)`, ce qui peut se traduire par « si tu es majeur ». Les tests sur des booléens sont donc faciles à lire et à comprendre, pour peu que vous ayez donné des noms clairs à vos variables comme je vous ai dit de le faire dès le début.

Tenez, voici un autre test imaginaire :

c

```c
if (majeur && garcon)
```

Ce test signifie « si tu es majeur ET que tu es un garçon ». `garcon` est ici une autre variable booléenne qui vaut 1 si vous êtes un garçon, et 0 si vous êtes... une fille ! Bravo, vous avez tout compris !

Les booléens permettent donc de dire si quelque chose est vrai ou faux.
C'est vraiment utile et ce que je viens de vous expliquer vous permettra de comprendre bon nombre de choses par la suite.

> Petite question : si on fait le test if `(majeur == 1)`, ça marche aussi, non ?

Tout à fait. Mais le principe des booléens c'est justement de raccourcir l'expression du `if` et de la rendre plus facilement lisible. Avouez que `if (majeur)` ça se comprend très bien, non ?

Retenez donc : si votre variable est censée contenir un nombre (comme un âge), faites un test sous la forme `if (variable == 1)`.
Si au contraire votre variable est censée contenir un booléen (c'est-à-dire soit 1 soit 0 pour dire vrai ou faux), faites un test sous la forme `if (variable)`.

La condition switch

La condition `if... else` que l'on vient de voir est le type de condition le plus souvent utilisé.
En fait, il n'y a pas 36 façons de faire une condition en C. Le `if... else` permet de gérer tous les cas.

Toutefois, le `if... else` peut s'avérer quelque peu... répétitif. Prenons cet exemple :

c

```c
if (age == 2)
{
    printf("Salut bebe !");
}
else if (age == 6)
{
    printf("Salut gamin !");
}
else if (age == 12)
{
    printf("Salut jeune !");
}
else if (age == 16)
{
    printf("Salut ado !");
}
else if (age == 18)
{
    printf("Salut adulte !");
}
else if (age == 68)
{
    printf("Salut papy !");
}
else
{
    printf("Je n'ai aucune phrase de prete pour ton age");
}
```

Construire un switch

Les informaticiens détestent faire des choses répétitives, on a eu l'occasion de le vérifier plus tôt.

Alors, pour éviter d'avoir à faire des répétitions comme ça quand on teste la valeur d'une seule et même variable, ils ont inventé une autre structure que le `if... else`. Cette structure particulière s'appelle `switch`. Voici un `switch` basé sur l'exemple qu'on vient de voir :

c

```c
switch (age)
{
case 2:
  printf("Salut bebe !");
  break;
case 6:
  printf("Salut gamin !");
  break;
case 12:
  printf("Salut jeune !");
  break;
case 16:
  printf("Salut ado !");
  break;
case 18:
  printf("Salut adulte !");
  break;
case 68:
  printf("Salut papy !");
  break;
default:
  printf("Je n'ai aucune phrase de prete pour ton age  ");
  break;
}
```

Imprégnez-vous de mon exemple pour créer vos propres `switch`. On les utilise plus rarement, mais c'est vrai que c'est pratique car ça fait (un peu) moins de code à taper.

L'idée c'est donc d'écrire `switch (maVariable)` pour dire « je vais tester la valeur de la variable `maVariable` ». Vous ouvrez ensuite des accolades que vous refermez tout en bas.

Ensuite, à l'intérieur de ces accolades, vous gérez tous les cas : `case 2`, `case 4`, `case 5`, `case 45`...

> Vous devez mettre une instruction `break;` obligatoirement à la fin de chaque cas. Si vous ne le faites pas, alors l'ordinateur ira lire les instructions en dessous censées être réservées aux autres cas ! L'instruction `break;` commande en fait à l'ordinateur de « sortir » des accolades.

Enfin, le cas `default` correspond en fait au `else` qu'on connaît bien maintenant. Si la variable ne vaut aucune des valeurs précédentes, l'ordinateur ira lire le `default`.

Gérer un menu avec un `switch`

Le `switch` est très souvent utilisé pour faire des menus en console.

Je crois que le moment est venu de pratiquer un peu !

Au boulot !

En console, pour faire un menu, on fait des `printf` qui affichent les différentes options possibles.

Chaque option est numérotée, et l'utilisateur doit entrer le numéro du menu qui l'intéresse.

Voici par exemple ce que la console devra afficher :

```
=== Menu ===
1. Royal Cheese
2. Mc Deluxe
3. Mc Bacon
4. Big Mac
Votre choix ?
```

Voici votre mission (si vous l'acceptez) : reproduisez ce menu à l'aide de `printf` (facile), ajoutez un `scanf` pour enregistrer le choix de l'utilisateur dans une variable `choixMenu`, et enfin faites un `switch` pour dire à l'utilisateur « tu as choisi le menu Royal Cheese » par exemple.

Allez, au travail !

Correction

Voici la solution (j'espère que vous l'avez trouvée !) :

c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int choixMenu;

  printf("=== Menu ===\n\n");
  printf("1. Royal Cheese\n");
  printf("2. Mc Deluxe\n");
  printf("3. Mc Bacon\n");
  printf("4. Big Mac\n");
  printf("\nVotre choix ? ");
  scanf("%d", &choixMenu);

  printf("\n");

  switch (choixMenu)
  {
    case 1:
       printf("Vous avez choisi le Royal Cheese. Bon choix !");
       break;
```

```
23      case 2:
24         printf("Vous avez choisi le Mc Deluxe. Berk, trop de sauce...");
25         break;
26      case 3:
27         printf("Vous avez choisi le Mc Bacon. Bon, ca passe encore ca ;o)");
28         break;
29      case 4:
30         printf("Vous avez choisi le Big Mac. Vous devez avoir tres faim !");
31         break;
32      default:
33         printf("Vous n'avez pas rentre un nombre correct. Vous ne mangerez rien du tout !");
34         break;
35   }
36
37   printf("\n\n");
38
39   return 0;
40 }
```

Et voilà le travail !

J'espère que vous n'avez pas oublié le `default` à la fin du `switch` ! En effet, quand vous programmez vous devez toujours penser à tous les cas. Vous avez beau dire de taper un nombre entre 1 et 4, vous trouverez toujours un imbécile qui ira taper `10` ou encore `Salut` alors que ce n'est pas ce que vous attendez.

Bref, soyez toujours vigilants de ce côté-ci : ne faites pas confiance à l'utilisateur, il peut parfois entrer n'importe quoi. Prévoyez toujours un cas `default` ou un `else` si vous faites ça avec des `if` .

> Je vous conseille de vous familiariser avec le fonctionnement des menus en console, car on en fait souvent dans des programmes console et vous en aurez sûrement besoin.

Les ternaires : des conditions condensées

Il existe une troisième façon de faire des conditions, plus rare.

On appelle cela des **expressions ternaires**.
Concrètement, c'est comme un `if... else` , sauf qu'on fait tout tenir sur une seule ligne !

Comme un exemple vaut mieux qu'un long discours, je vais vous donner deux fois la même condition : la première avec un `if... else` , et la seconde, identique, mais sous forme d'une expression ternaire.

Une condition `if... else` bien connue

Supposons qu'on ait une variable booléenne `majeur` qui vaut vrai (1) si on est majeur, et faux (0) si on est mineur.
On veut changer la valeur de la variable `age` en fonction du booléen, pour mettre "18" si on est

majeur, "17" si on est mineur. C'est un exemple complètement stupide je suis d'accord, mais ça me permet de vous montrer comment on peut se servir des expressions ternaires.

Voici comment faire cela avec un `if... else` :

c

```c
if (majeur)
  age = 18;
else
  age = 17;
```

> Notez que j'ai enlevé dans cet exemple les accolades car elles sont facultatives s'il n'y a qu'une instruction, comme je vous l'ai expliqué plus tôt.

La même condition en ternaire

Voici un code qui fait exactement la même chose que le code précédent, mais écrit cette fois sous forme ternaire :

c

```c
age = (majeur) ? 18 : 17;
```

Les ternaires permettent, sur une seule ligne, de changer la valeur d'une variable en fonction d'une condition. Ici la condition est tout simplement `majeur`, mais ça pourrait être n'importe quelle condition plus longue bien entendu. Un autre exemple ? `autorisation = (age >= 18) ? 1 : 0;`

Le point d'interrogation permet de dire « est-ce que tu es majeur ? ». Si oui, alors on met la valeur 18 dans `age`. Sinon (le deux-points `:` signifie `else` ici), on met la valeur 17.

Les ternaires ne sont pas du tout indispensables, personnellement je les utilise peu car ils peuvent rendre la lecture d'un code source un peu difficile. Ceci étant, il vaut mieux que vous les connaissiez pour le jour où vous tomberez sur un code plein de ternaires dans tous les sens !

En résumé

- Les **conditions** sont à la base de tous les programmes. C'est un moyen pour l'ordinateur de **prendre une décision** en fonction de la valeur d'une variable.
- Les mots-clés `if`, `else if`, `else` signifient respectivement « si », « sinon si », « sinon ». On peut écrire autant de `else if` que l'on désire.
- Un **booléen** est une variable qui peut avoir deux états : vrai (1) ou faux (0) (toute valeur différente de 0 est en fait considérée comme « vraie »). On utilise des `int` pour stocker des booléens car ce ne sont en fait rien d'autre que des nombres.
- Le `switch` est une alternative au `if` quand il s'agit d'analyser la valeur d'une variable. Il permet de rendre un code source plus clair si vous vous apprêtiez à tester de nombreux cas. Si vous

utilisez de nombreux `else if` c'est en général le signe qu'un `switch` serait plus adapté pour rendre le code source plus lisible.

- Les **ternaires** sont des conditions très concises qui permettent d'affecter rapidement une valeur à une variable en fonction du résultat d'un test. On les utilise avec parcimonie car le code source a tendance à devenir moins lisible avec elles.

Que pensez-vous de ce cours ?

UNE BÊTE DE CALCUL

LES BOUCLES

Le professeur

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Découvrez aussi ce cours en...

Livre PDF

OpenClassrooms

L'entreprise

Alternance

Forum

Blog

Nous rejoindre

Entreprises

Business

En plus