JAVASCRIPT 2 - Conditionals, Loops, Fonctions

The if Statement

```
if (condition) {
  statements
}
```

L'instruction if

Très souvent, lorsque vous écrivez du code, vous souhaitez effectuer différentes actions en fonction de différentes conditions.
Vous pouvez le faire en utilisant des instructions conditionnelles dans votre code.

Utilisez if pour spécifier un bloc de code qui sera exécuté si une condition spécifiée est vraie.

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 < myNum2) {
  alert("JavaScript is easy to learn.");
}
```

Les instructions ne seront exécutées que si la condition spécifiée est vraie.

Exemple:

Result:

Resultat

L'instruction if

C'est un autre exemple d'une fausse déclaration conditionnelle.

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
  alert("JavaScript is easy to learn.");
}
```

Lorsque la condition est évaluée à false, l'instruction d'alerte est ignorée et le programme continue avec la ligne après l'accolade de fermeture de l'instruction if.

Notez que si est en minuscules. Les lettres majuscules (If ou IF) générent une erreur.

The if else Statement

```
if (expression) {
  // executed if condition is true
}
else {
  // executed if condition is false
}
```

L'autre déclaration
Utilisez l'instruction else pour spécifier un bloc de code qui s'exécutera si la condition est fausse.

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
  alert("This is my first condition");
}
else {
  alert("This is my second condition");
}
```

L'exemple ci-dessus illustre l'utilisation d'une instruction if ... sinon. L'exemple ci-dessus dit:

- Si myNum1 est plus grand que myNum2, alerte "Ceci est ma première condition";
- Sinon, alerte "C'est ma deuxième condition".

Le navigateur imprimera la deuxième condition, car 7 n'est pas supérieur à 10.

The if else if else Statement

```
var course = 5;
if (course == 3) {
  document.write("<h1>HTML Tutorial</h1>");
} else if (course == 5) {
  document.write("<h1>CSS Tutorial</h1>");
} else {
  document.write("<h1>JavaScript Tutorial</h1>");
}
```

Le code ci-dessus indique:

- si cours est égal à 3, sortie "Tutoriel HTML";
- sinon, si cours est égal à 2, sortie "Didacticiel CSS";
- si aucune des conditions ci-dessus n'est vraie, sortez "Didacticiel JavaScript";

Cours est égal à 1, donc nous obtenons le résultat suivant:

HTML Tutorial

La dernière instruction else "termine" l'instruction if else et devrait toujours être écrite après les instructions if et else if.

Vous pouvez écrire autant d'instructions if si vous en avez besoin.

Vous pouvez avoir autant de déclarations de cas que nécessaire.

```
var course = 5;
if (course == 3) {
  document.write("<h1>HTML Tutorial</h1>");
} else if (course == 5) {
  document.write("<h1>CSS Tutorial</h1>");
} else {
  document.write("<h1>JavaScript Tutorial</h1>");
}
```

Le dernier bloc sera exécuté quand aucune des conditions n'est vraie.
Changeons la valeur de la variable de cours dans notre exemple précédent.

The result:

JavaScript Tutorial

The Switch Statement

```
switch (expression) {
  case x1:
    statements
    break;
  case x2:
    statements
    break;
  default:
    statements
}
```

L'expression du commutateur est évaluée une fois.
La valeur de l'expression est comparée avec les valeurs de chaque cas. S'il y a une correspondance, le bloc de code associé est exécuté.

Vous pouvez obtenir le même résultat avec plusieurs instructions if ... else, mais l'instruction switch est plus efficace dans de telles situations.

```
var day = 2;
switch (day) {
  case 1:
    document.write("Monday");
    break;
  case 2:
    document.write("Tuesday");
    break;
  case 3:
    document.write("Wednesday");
  break;
  default:
    document.write("Another day");
}

// Outputs "Tuesday"
```

Exemple.

Dans les cas où vous avez besoin de tester pour de multiples conditions, écriture des déclarations else pour chaque condition pourrait ne pas être la meilleure solution.
L'instruction switch est utilisée pour effectuer différentes actions en fonction de différentes conditions.

```
var color = "yellow";
switch (color) {
  case "blue":
    document.write("This is blue.");
    break;
  case "red":
    document.write("This is red.");
    break;
  case "green":
    document.write("This is green.");
    break;
  case "orange":
    document.write("This is orange.");
    break;
  default:
    document.write("Color not found.");
}
```

Le bloc par défaut peut être omis, s'il n'est pas nécessaire de traiter le cas où aucune correspondance n'est trouvée.

Le Mot-clé Break

Lorsque le code JavaScript atteint un mot clé Break, il se détache du bloc de commutateur.
Cela arrêtera l'exécution de plus de code et de tests de cas à l'intérieur du bloc. Habituellement, une pause doit être mise dans chaque déclaration de cas.

Le Mot-clé Default

Le mot clé Default spécifie le code à exécuter s'il n'y a pas de correspondance de cas.

The For Loop

Boucles

Les boucles peuvent exécuter un bloc de code un certain nombre de fois. Ils sont pratiques dans les cas où vous souhaitez exécuter le même code à plusieurs reprises, en ajoutant une valeur différente à chaque fois.

JavaScript comporte trois types de boucles: for, while et do while.

La boucle for est couramment utilisée lors de la création d'une boucle.

```
for (statement 1; statement 2; statement 3) {
  code block to be executed
}
```

La syntaxe:

L'instruction 1 est exécutée avant que la boucle (le bloc de code) démarre.
L'instruction 2 définit la condition d'exécution de la boucle (le bloc de code).
L'instruction 3 est exécutée chaque fois que la boucle (le bloc de code) a été exécutée.

Comme vous pouvez le voir, une boucle classique a trois composantes, ou des énoncés.

```
for (i=1; i<=5; i++) {
  document.write(i + "<br />");
}
```

L'exemple ci-dessous crée une boucle for qui imprime les numéros 1 à 5.

Result:

```
1
2
3
4
5
```

Dans cet exemple, l'instruction 1 définit une variable avant le début de la boucle (var i = 1).
L'énoncé 2 définit la condition de la boucle à exécuter (i doit être inférieur ou égal à 5).
L'instruction 3 augmente une valeur (i ++) chaque fois que le bloc de code dans la boucle a été exécuté.

```
var i = 1;
for (; i<=5; i++) {
  document.write(i + "<br />");
}
```

L'instruction 1 est facultative et peut être omise si vos valeurs sont définies avant le début de la boucle.

```
for (i=1, text=""; i<=5; i++) {
  text = i;
  document.write(i + "<br />");
}
```

En outre, vous pouvez initier plus d'une valeur dans l'instruction 1, en utilisant des virgules pour les séparer.

```
var i = 0;
for (; i < 10; ) {
  document.write(i);
  i++;
}
```

L'instruction 3 permet de modifier la variable initiale. Il peut faire n'importe quoi, y compris incrément négatif (i--), incrément positif (i = i + 15), ou autre chose.
L'instruction 3 est également facultative et peut être omise si vous augmentez vos valeurs à l'intérieur de la boucle comme ci-dessus.

Si l'instruction 2 renvoie true, la boucle recommencera à nouveau, si elle retourne false, la boucle se terminera.
L'énoncé 2 est également facultatif.
Si vous omettez l'instruction 2, vous devez fournir une pause dans la boucle. Sinon, la boucle ne finira jamais.

The While Loop

```
while (condition) {
  code block
}
```

La boucle while se répète à travers un bloc de code, tant qu'une condition spécifiée est vraie. La condition peut être toute instruction conditionnelle qui renvoie true ou false.

```
var i=0;
while (i<=10) {
  document.write(i + "<br />");
  i=i+1
}
```

Prenons l'exemple suivant.

```
0
1
2
3
4
5
```

La boucle continuera à fonctionner tant que i est inférieur ou égal à 10. Chaque fois que la boucle s'exécute, je vais augmenter de 1.

Cela affichera les valeurs de 0 à 10.

Si vous oubliez d'augmenter la variable utilisée dans la condition, la boucle ne finira jamais. Assurez-vous que la condition dans une boucle while devient finalement fausse.

The Do...While Loop

```
do {
  code block
}
while (condition);
```

La boucle do ... while est une variante de la boucle while. Cette boucle exécute le bloc de code une fois, avant de vérifier si la condition est vraie, puis elle répétera la boucle tant que la condition est vraie.

```
var i=20;
do {
  document.write(i + "<br />");
  i=i-1
}
while (i>=20);
```

Exemple

Notez le point-virgule utilisé à la fin de la boucle do ... while.

```
20
21
22
23
24
25
```

La boucle sera toujours exécutée au moins une fois, même si la condition est fausse, car le bloc de code est exécuté avant que la condition ne soit testée.

Break and Continue

```
for (i = 0; i <= 10; i++) {
  if (i == 3) {
    break;
  }
  document.write(i + "<br />");
}
```

L'instruction break "saute" d'une boucle et continue à exécuter le code après la boucle.

```
0
1
2
```

Une fois que j'atteindrai 5, il sortira de la boucle.

User-Defined Functions

Une fonction JavaScript est un bloc de code conçu pour exécuter une tâche particulière.
Les principaux avantages de l'utilisation des fonctions:
Réutilisation du code: Définissez le code une fois et utilisez-le plusieurs fois.
Utilisez le même code plusieurs fois avec des arguments différents, pour produire des résultats différents.

A JavaScript function is executed when "something" invokes, or calls, it.

```
function name() {
  //code to be executed
}
```

Pour définir une fonction JavaScript, utilisez le mot-clé function, suivi d'un nom suivi d'un ensemble de parenthèses ().

Le code à exécuter par la fonction est placé entre crochets {}. Les noms de fonctions peuvent contenir des lettres, des chiffres, des caractères de soulignement et des signes de dollar (les mêmes règles que les variables).

```
function myFunction() {
  alert("calling a function");
}

myFunction();
```

Pour exécuter la fonction, vous devez l'appeler.
Pour appeler une fonction, commencez par le nom de la fonction, puis suivez-la avec les arguments entre parenthèses.

N'oubliez pas de terminer l'instruction avec un point-virgule après avoir appelé la fonction.

```
function myFunction() {
  alert("Alert box!");
}

myFunction();

// some other code

myFunction();
```

Une fois la fonction définie, JavaScript vous permet de l'appeler autant de fois que vous le souhaitez.

Functions Parameters

```
functionName(param1, param2, param3) {
  // some code
}
```

Les fonctions peuvent prendre des paramètres.
Les paramètres de fonction sont les noms répertoriés dans la définition de la fonction. Comme pour les variables, les paramètres doivent être donnés des noms, qui sont séparés par des virgules dans les parenthèses.

```
function sayHello(name) {
  alert("Hi, " + name);
}

sayHello("David");
```

Après avoir défini les paramètres, vous pouvez les utiliser dans la fonction. Cette fonction prend un paramètre, appelé nom. Lorsque vous appelez la fonction, donnez la valeur du paramètre (argument) dans les parenthèses.

```
function sayHello(name) {
  alert("Hi, " + name);
}

sayHello("David");
sayHello("Sarah");
sayHello("John");
```

Vous pouvez définir une seule fonction et lui transmettre différentes valeurs de paramètre (arguments).

Les arguments de fonction sont les valeurs réelles transmises à (et reçues par) la fonction.

Cela exécute le code de la fonction à chaque fois pour l'argument fourni.

Multiple Parameters

```
function myFunc(x, y) {
  // some code
}
```

Vous pouvez définir plusieurs paramètres pour une fonction en les séparant par des virgules. L'exemple ci-dessus définit la fonction myFunc pour prendre deux paramètres.

```
function sayHello(name, age) {
  document.write( name + " is " + age + " years old.");
}
```

Les paramètres sont utilisés dans la définition de la fonction.

```
function sayHello(name, age) {
  document.write( name + " is " + age + " years old.");
}

sayHello("John", 20)
```

Lorsque vous appelez la fonction, fournissez les arguments dans le même ordre dans lequel vous les avez définis.

Si une fonction est appelée avec des arguments manquants (moins que déclaré), les valeurs manquantes sont définies sur undefined, ce qui indique qu'une variable n'a pas reçu de valeur.

Après avoir défini la fonction, vous pouvez l'appeler autant de fois que nécessaire.
Les fonctions JavaScript ne vérifient pas le nombre d'arguments reçus.

The return statement

Une fonction peut avoir une instruction de retour facultative. Il est utilisé pour renvoyer une valeur de la fonction.

Cette déclaration est utile lorsque vous effectuez des calculs nécessitant un résultat.

Lorsque JavaScript atteint une instruction de retour, la fonction cesse d'être exécutée.

```
function myFunction(x, y) {
  return a * b;
}

var v = myFunction(5, 8);
document.write(v);
```

Utilisez l'instruction return pour retourner une valeur.

Par exemple, calculons le produit de deux nombres et renvoyons le résultat.

Si vous ne renvoyez rien d'une fonction, il retournera indéfini.

```
function addNumbers(a, b) {
  var c = a+b;
  return c;
}
document.write( addNumbers(40, 2) );
```

Un autre exemple: //Outputs 42

La commande document.write génère la valeur renvoyée par la fonction, qui est la somme des deux paramètres.

Alert, Prompt, Confirm

```
alert("Do you really want to leave this page?");
```

JavaScript propose trois types de boîtes contextuelles: les zones Alerte, Demander et Confirmer.

Boîte d'alerte
Une zone d'alerte est utilisée lorsque vous souhaitez vous assurer que l'information est transmise à l'utilisateur.
Lorsqu'une boîte d'alerte apparaît, l'utilisateur doit cliquer sur OK pour continuer.
La fonction d'alerte prend un seul paramètre, qui est le texte affiché dans la boîte popup.

Result:

Pour afficher des sauts de ligne dans une zone contextuelle, utilisez une barre oblique inverse suivie du caractère n.

```
alert("Hello\nHow are you?");
```

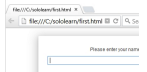
Soyez prudent lors de l'utilisation des boîtes d'alerte, car l'utilisateur peut continuer à utiliser la page uniquement après avoir cliqué sur OK.

```
var user = prompt("Please enter your name");
alert(user);
```

Une boîte d'invite est souvent utilisée pour que l'utilisateur saisisse une valeur avant d'entrer une page.
Lorsqu'une boîte d'invite s'affiche, l'utilisateur doit cliquer sur OK ou sur Annuler pour continuer après avoir saisi la valeur d'entrée.
Si l'utilisateur clique sur OK, la zone retourne la valeur d'entrée. Si l'utilisateur clique sur Annuler, la zone renvoie null.

La méthode prompt() prend deux paramètres.

- Le premier est le libellé que vous souhaitez afficher dans la zone de texte.
- La seconde est une chaîne par défaut à afficher dans la zone de texte (facultatif).

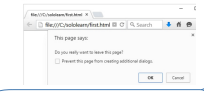


Resultat

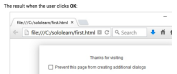
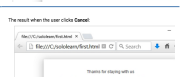
```
var result = confirm("Do you really want to leave this page?");
if (result == true) {
  alert("Thanks for visiting");
}
else {
  alert("Thanks for staying with us");
}
```

Une case de confirmation est souvent utilisée pour que l'utilisateur vérifie ou accepte quelque chose.
Lorsqu'une boîte de confirmation apparaît, l'utilisateur doit cliquer sur OK ou sur Annuler pour continuer.
Si l'utilisateur clique sur OK, la zone retourne true. Si l'utilisateur clique sur Annuler, la zone renvoie false.

Result:



The result when the user clicks **OK**:



The result when the user clicks **Cancel**: