

## Programmation langage objet

### Création d'une page html

Site de référence :

<http://fr.openclassrooms.com/informatique/cours/dynamisez-vos-sites-web-avec-javascript>

## 1. Rappel structure d'une page html

### 1.1. Les balises

```
<!DOCTYPE html>
2<html>
3  <head>
4  <link rel="icon" type="image/png" href="icone.png" />
5    <meta charset="utf-8" />
6    <title>Titre</title>
7  </head>
8
9  <body>
10
11 </body>
12</html>
```

## 2. Le langage CSS

Les **CSS**, *Cascading Style Sheets* (feuilles de styles en cascade), servent à mettre en forme des documents web, type page HTML ou XML, par l'intermédiaire de propriétés d'apparence (couleurs, bordures, polices, etc.) et de placement (largeur, hauteur, côte à côte, dessus-dessous, etc.). Le rendu d'une page web peut être intégralement modifié sans aucun code supplémentaire dans la page web.

On peut écrire du code en langage CSS à trois endroits différents :

- dans un fichier `.css` (*méthode la plus recommandée*) ;
- dans l'en-tête `<head>` du fichier HTML ;
- directement dans les balises du fichier HTML *via* un attribut `style` (*méthode la moins recommandée*). (Ex : `<div align="center">`)

C'est le contenu, `<link rel="stylesheet" href="style.css" />` : qui indique que le fichier HTML est associé à un fichier appelé `style.css` qui est chargé de la mise en forme.

Exemple : on veut que le texte du document html soit de couleur rouge

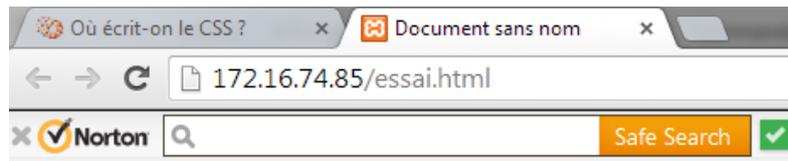
Dans le fichier html on mettra :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<link rel="stylesheet" href="fichiercss.css" />
<title>Document sans nom</title>
</head>
<body >
  <p>couleur rouge</p>
</body>
</html>
```

Dans le fichier css on mettra :

```
@charset "utf-8";  
/* CSS Document */  
p  
{  
color: red;  
}
```

On obtient :



couleur rouge

### 3. Le langage javascript

#### 3.1. Présentation

Le JavaScript (souvent abrégé JS) est un langage de programmation de scripts principalement utilisé dans les pages web interactives mais aussi côté serveur. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets .

#### 3.2. Introduction

```
<!doctype html>  
<html>  
<head>  
<meta charset="utf-8">  
<title>cours javascript</title></head>  
<body>  
  
<script type="text/javascript">  
alert('bonjour sur le cours java');  
</script>  
  
</body>  
</html>
```

Code javascript



Dans le code HTML donné précédemment, on remarque quelques nouveautés.

Tout d'abord, un élément `<script>` est présent : c'est lui qui contient le code Javascript que voici :

```
alert('bonjour sur le cours java');
```

Il s'agit d'une instruction, c'est-à-dire une commande, un ordre, ou plutôt une action que l'ordinateur va devoir réaliser. Les langages de programmation sont constitués d'une suite d'instructions qui, mises bout à bout, permettent d'obtenir un programme ou un script complet.

Dans cet exemple, il n'y a qu'une instruction : l'appel de la fonction `alert()`.

`alert()` est une instruction simple, appelée **fonction**, qui permet d'afficher une boîte de dialogue contenant un message. Ce message est placé entre apostrophes, elles-mêmes placées entre les parenthèses de la fonction `alert()`.

### 3.3. Mise en place d'un script

Il existe deux méthodes pour intégrer du java script dans une page html :

- Dans la page (présentation ci-dessus)
- Externe : on fait appel à une page ayant l'extension `.js`

Ce fichier est ensuite appelé depuis la page Web au moyen de l'élément `<script>` et de son attribut `src` qui contient l'URL du fichier `.js`. Voici un exemple :

```
<body>  
<script src="http://mrdoob.github.com/three.js/build/three.min.js"></script>
```

### 3.4. Les instructions

La syntaxe du Javascript n'est pas compliquée. De manière générale, les instructions doivent être séparées par un point-virgule que l'on place à la fin de chaque instruction :

```
Instruction1 ;  
Instruction2 ;
```

### 3.5. Les variables

Pour faire simple, une variable est un espace de stockage sur votre ordinateur permettant d'enregistrer tout type de donnée, que ce soit une chaîne de caractères, une valeur numérique ou bien des structures un peu plus particulières.

- Déclaration d'une variable

Déclarer une variable sert à lui réserver un espace de stockage en mémoire. Une fois la variable déclarée, vous pouvez commencer à y stocker des données sans problème.

Le nom d'une variable ne peut contenir que des caractères alphanumériques, autrement dit les lettres de A à Z et les chiffres de 0 à 9 ; l'underscore (`_`) et le dollar (`$`) sont aussi acceptés. Autre chose : le nom de la variable ne peut pas commencer par un chiffre et ne peut pas être constitué uniquement de mots-clés utilisés par le Javascript.

Pour déclarer une variable, il vous suffit d'écrire la ligne suivante :

```
var myVariable;
```

**Le Javascript étant un langage sensible à la casse, faites bien attention à ne pas vous tromper sur les majuscules et minuscules.**

Le mot-clé `var` est présent pour indiquer que vous déclarez une variable. Une fois celle-ci déclarée, il ne vous est plus nécessaire d'utiliser ce mot-clé pour cette variable et vous pouvez y stocker ce que vous souhaitez :

```
var myVariable;  
myVariable = 2;
```

On peut aussi simplifier par :

```
var myVariable = 2;
```

Le signe = sert à attribuer une valeur à la variable. Quand on donne une valeur à une variable, on dit que l'on fait une **affectation**, car on affecte une valeur à la variable.

Lorsqu'il y a plusieurs variables à déclarer, il est possible de les déclarer en même temps.

```
var renderer, scene, camera, mesh, sphere;
```

- Les types de variables

Contrairement à de nombreux langages, le Javascript est un langage typé *dynamiquement*. Cela veut dire, que toute déclaration de variable se fait avec le mot-clé var sans distinction du contenu. On peut y mettre n'importe quel type de données numérique ou texte.

- Le type numérique : il représente tout nombre, que ce soit un entier, un négatif, un nombre scientifique, etc.:

```
var number = 5;
```

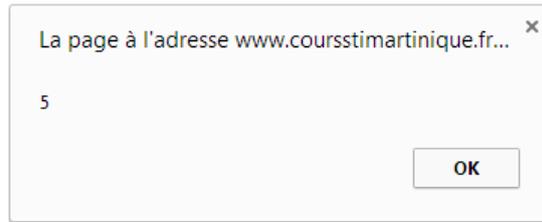
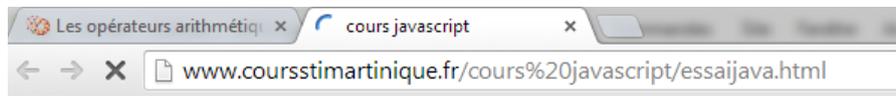
- Le type texte : On peut l'assigner de deux façons différentes :

```
var text1 = "Mon premier texte"; // Avec des guillemets  
var text2 = 'Mon deuxième texte'; // Avec des apostrophes
```

### 3.6. Les calculs

Dans une programmation, on a besoin des fois de faire des calculs. Faire des calculs en programmation est quasiment tout aussi simple que sur une calculatrice, exemple :

```
<!doctype html>  
<html>  
<head>  
<meta charset="utf-8">  
<title>cours javascript</title>  
</head>  
<body>  
  
<script type="text/javascript">  
var result = 3 + 2;  
alert(result); // Affiche : « 5 »  
  
</script>  
  
</body>  
</html>
```



Avec deux variables contenant elles-mêmes des nombres :

```
var number1 = 3, number2 = 2, result;  
result = number1 * number2;  
alert(result); // Affiche : « 6 »
```

- Simplification d'un calcul :

Lors de certain programme nous avons besoin d'écrire une incrémentation

```
var number = 3;  
number = number + 5;  
alert(number); // Affiche : « 8 »
```

Ce n'est pas spécialement long ou compliqué à faire, mais cela peut devenir très vite rébarbatif, il existe donc une solution plus simple pour incrémenter :

```
var number = 3;  
number += 5;  
alert(number); // Affiche : « 8 »
```

Ce code a exactement le même effet que le précédent mais est plus rapide à écrire.

#### Remarque :

On peut aussi utiliser le signe + lorsqu'on veut ajouter du texte (concaténation)

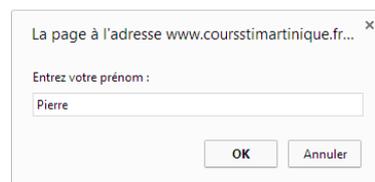
```
var hi = 'cours', name = 'sti2d', result;  
result = hi + name;  
alert(result); // Affiche : « courssti2d »
```

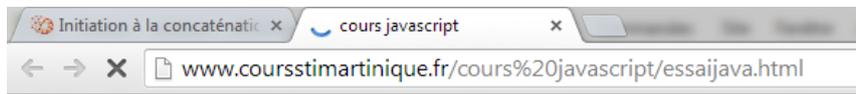
### 3.7. Inter action avec l'utilisateur

- La fonction prompt().

Exemple :

```
var userName = prompt('Entrez votre prénom :');  
alert(userName); // Affiche le prénom entré par l'utilisateur
```





La fonction `prompt()` s'utilise comme `alert()` mais a une petite particularité. Elle renvoie ce que l'utilisateur a écrit sous forme d'une chaîne de caractères.

**Attention :**

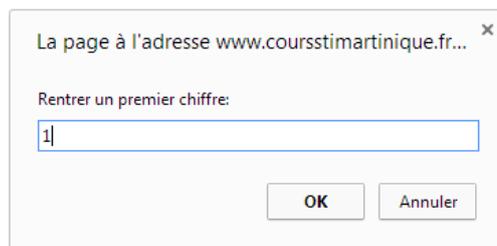
Tout ce qui est écrit dans le champ de texte de `prompt()` est récupéré sous forme d'une chaîne de caractères, que ce soit un chiffre ou non. Du coup, si vous utilisez l'opérateur `+`, vous ne ferez pas une addition mais une concaténation. Pour résoudre le problème, il suffit de convertir la chaîne de caractères en nombre. Pour cela, vous allez avoir besoin de la fonction `parseInt()` qui s'utilise de cette manière :

```
var text = '1337', number;  
number = parseInt(text);  
alert(typeof number); // Affiche : « number »  
alert(number); // Affiche : « 1337 »
```

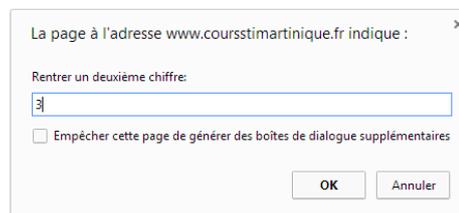
`typeof` indique le type de la variable. Dans notre cas « number »

Réaliser l'exercice suivant :

- Rentrer un premier chiffre



- Rentrer un deuxième chiffre



- Additionner les deux chiffres
- Afficher le résultat avec un texte



Correction :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>cours javascript</title>
</head>
<body>

<script type="text/javascript">
var chiffre1 = prompt('Rentrez un premier chiffre:');
var chiffre2 = prompt('Rentrez un deuxième chiffre:');
var resultat=parseInt(chiffre1)+parseInt(chiffre2);
alert('le résultat est égal à '+resultat); // Afficher le résultat »
</script>

</body>
</html>
```

### 3.8. Les conditions

#### 3.8.1.La structure if pour dire « si »

On l'utilise lorsqu'on veut contrôler quelque chose  
un exemple très simple :

```
if (true) {
alert("Ce message s'est bien affiché.");
}
if (false) {
alert("Pas la peine d'insister, ce message ne s'affichera pas.");
}
```

#### 3.8.2.De quoi est constituée une condition :

- De la structure conditionnelle if ;
- De parenthèses qui contiennent la condition à analyser, ou plus précisément le booléen retourné par les opérateurs conditionnels ;

- D'accolades qui permettent de définir la portion de code qui sera exécutée si la condition se vérifie. À noter que nous plaçons ici la première accolade à la fin de la première ligne de condition, mais vous pouvez très bien la placer comme vous le souhaitez (en dessous, par exemple).

**Le code d'une condition est exécuté si le booléen reçu est true alors que false empêche l'exécution du code.**

### 3.8.3. Les opérateurs de conditions

Comme leur nom l'indique, ces opérateurs vont permettre de comparer diverses valeurs entre elles. En tout, ils sont au nombre de huit, les voici :

Opérateur	Signification
==	égal à
!=	différent de
===	contenu et type égal à
!==	contenu ou type différent de
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

### 3.8.4. Les opérateurs logiques

Ils fonctionnent sur le même principe qu'une table de vérité en électronique. Ils sont nombre de trois :

Opérateur	Type de logique	Utilisation
&&	ET	valeur1 && valeur2
	OU	valeur1    valeur2
!	NON	!valeur

Exercice :

On veut contrôler si deux valeurs sont égales :

- Rentrer un premier chiffre



La page à l'adresse www.coursstimartinique.fr... ✕

Rentrer un premier chiffre:

- Rentrer un deuxième chiffre



- Afficher le résultat si les chiffres sont égaux ou pas égaux



Correction :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>cours javascript1</title>
</head>
<body>

<script type="text/javascript">
var chiffre1 = prompt('Rentrer un premier chiffre:');
var chiffre2 = prompt('Rentrer un deuxième chiffre:');
if (chiffre1 != chiffre2) {
alert("Les deux chiffres ne sont pas égaux.");
}
if (chiffre1 = chiffre2) {
alert("Les deux chiffres sont égaux.");
}

</script>

</body>
</html>
```

### 3.8.5. la fonction confirm()

On lui passe en paramètre une chaîne de caractères qui sera affichée à l'écran et elle retourne un booléen en fonction de l'action de l'utilisateur ; vous allez comprendre en essayant :

Exemple :

```
if (confirm('Voulez-vous exécuter le code Javascript de cette page ?')) {  
  alert('Le code a bien été exécuté !');  
}
```



Je clique sur **OK**



Comme vous pouvez le constater, le code s'exécute lorsque vous cliquez sur le bouton OK et ne s'exécute pas lorsque vous cliquez sur Annuler. En clair : dans le premier cas la fonction renvoie true et dans le deuxième cas elle renvoie false.

### 3.8.6. La structure else pour dire « sinon »

Elle permet d'afficher un résultat dans le cas où la condition est fausse :

Exercice ci-dessus :

```
<!doctype html>  
<html>  
<head>  
<meta charset="utf-8">  
<title>cours javascript1</title>  
</head>  
<body>  
<script type="text/javascript">  
var chiffre1 = prompt('Rentrer un premier chiffre:');  
var chiffre2 = prompt('Rentrer un deuxième chiffre:');  
if (chiffre1 != chiffre2) {  
  alert("Les deux chiffres ne sont pas égaux.");  
}  
else {  
  alert("Les deux chiffres sont égaux.");  
}  
</script>  
</body>
```

### 3.8.7. La fonction switch

Prenons un exemple : nous avons un meuble avec quatre tiroirs contenant chacun des objets différents, et il faut que l'utilisateur puisse connaître le contenu du tiroir dont il entre le chiffre. Si nous voulions le faire avec if else ce serait assez long et fastidieux :

```
var drawer = parseInt(prompt('Choisissez le tiroir à ouvrir (1 à 4) :'));
switch (drawer) {
case 1:
alert('Contient divers outils pour dessiner : du papier, des crayons, etc. ');
break;
case 2:
alert('Contient du matériel informatique : des câbles, des composants, etc. ');
break;
case 3:
alert('Ah ? Ce tiroir est fermé à clé ! Dommage !');
break;
case 4:
alert('Contient des vêtements : des chemises, des pantalons, etc. ');
break;
default:
alert("Info du jour : le meuble ne contient que 4 tiroirs et, jusqu'à preuve du contraire, les tiroirs négatifs n'existent pas.");
}
```

Fonctionnement :

- On écrit le mot-clé switch suivi de la variable à analyser entre parenthèses et d'une paire d'accolades ;
- Dans les accolades se trouvent tous les cas de figure pour notre variable, définis par le mot-clé case suivi de la valeur qu'il doit prendre en compte (cela peut être un nombre mais aussi du texte) et de deux points ;
- Tout ce qui suit les deux points d'un case sera exécuté si la variable analysée par le switch contient la valeur du case ;
- À chaque fin d'un case on écrit l'instruction break pour « casser » le switch et ainsi éviter d'exécuter le reste du code qu'il contient ;
- Enfin on écrit le mot-clé default suivi de deux points. Le code qui suit cette instruction sera exécuté si aucun des cas précédents n'a été exécuté. Attention, cette partie est optionnelle, vous n'êtes pas obligés de l'intégrer à votre code.

#### 4. Les fonctions

- Caractéristiques :

```
function myFunction(arguments) {
// Le code que la fonction va devoir exécuter
}
```

Le mot-clé function est présent à chaque déclaration de fonction. C'est lui qui permet de dire « Voilà, j'écris ici une fonction ! » ;

Vient ensuite le nom de votre fonction, ici myFunction ;

S'ensuit un couple de parenthèses contenant ce que l'on appelle des **arguments**. Ces arguments servent à fournir des informations à la fonction lors de son exécution. Par exemple, avec la fonction alert() quand vous lui passez en paramètre ce que vous voulez afficher à l'écran ;

Et vient enfin un couple d'accolades contenant le code que votre fonction devra exécuter.

Il est important de préciser que tout code écrit dans une fonction ne s'exécutera que si vous appelez cette dernière (« appeler une fonction » signifie « exécuter »). Sans ça, le code qu'elle contient ne s'exécutera jamais.

Exemple : Exécution d'une fonction en cliquant sur un bouton

```
<script type="text/javascript">
function rouge(){
var couleurtexte='rouge';
alert(couleurtexte); // Affiche : « rouge»
}
</script>
<div id="bouton1"><button onclick="rouge();">fond rouge</button></div>
```

### **Attention :**

Toute variable déclarée dans une fonction n'est utilisable que dans cette même fonction ! Ces variables spécifiques à une seule fonction ont un nom : les variables locales.

#### 4.1.1. Créer et utiliser un argument

Les arguments sont des informations envoyées à une fonction. Ces informations peuvent servir à beaucoup de choses.

```
function myFunction(arg) {
    alert('Votre argument : ' + arg);
}
myFunction(prompt('Que souhaitez-vous passer en argument à la fonction ?'));
```

### **On peut utiliser des arguments multiples**

```
<script type="text/javascript">
function moar(first, second) {
    // On peut maintenant utiliser les variables « first » et « second » comme on le souhaite :
    alert('Votre premier argument : ' + first);
    alert('Votre deuxième argument : ' + second);
}
</script>
<div id="bouton2"><button onclick="moar(1,2)">Rentrer les valeurs</button></div>
</body>
```

## 5. Manipulation des propriétés CSS avec le javascript

Il y a deux manières de modifier le CSS d'un élément HTML, nous allons ici utiliser la méthode la plus simple et la plus utilisée : l'utilisation de la propriété style.

Pour accéder à la propriété style de notre élément nous allons utiliser la même manière que pour accéder à n'importe quelle propriété de notre élément :

On écrit d'abord « element.style » pour accéder à la propriété « style » de l'élément « element »

Une fois que l'on a accédé à sa propriété, on écrit en suivant le nom de la propriété et le paramètre.

Exemple :

```
element.style.width = '150px'; // On modifie la largeur de notre élément à 150px
```

### **Attention :**

En Javascript, les tirets sont interdits dans les noms des propriétés.

La solution est de supprimer les tirets et chaque mot suivant normalement un tiret voit sa première lettre devenir une majuscule.

Ainsi.

```
element.style.background-color = 'blue'; devient element.style.backgroundColor = 'blue';
```

Exercice :

- Changer la couleur de fond d'un texte en agissant sur deux boutons.
  - Un bouton pour la couleur rouge,
  - Un bouton pour la couleur bleu,



bonjour

fond rouge

fond bleu

Correction :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>cours javascript1</title>
<style type="text/css">

    #couleur {
        position: fixed;
        top: 10px;
        left: 10px;
        z-index:10;
        background-color:#FBOC10;
    }

    #bouton1 {
        position: fixed;
        top: 40px;
        left: 10px;
        z-index:10;
    }
    #bouton2
        {position:fixed;
        top:80px;
        left: 10px;
        z-index:10;
        }

</style>
</head>
<body>

<script type="text/javascript">
```

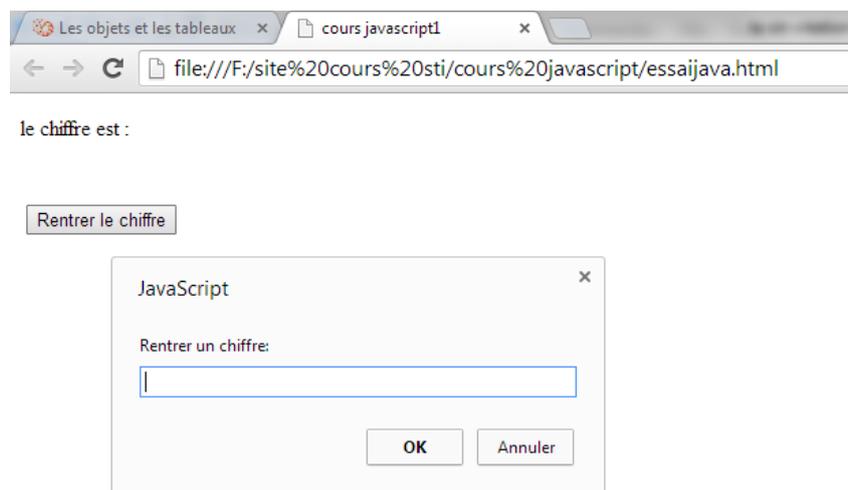
```

function rouge(){
couleur.style.backgroundColor='red';
}
function bleu(){
couleur.style.backgroundColor='blue';
}
</script>
<div id="couleur" >bonjour</div>
<div id="bouton1"><button onclick="rouge();">fond rouge</button></div>
<div id="bouton2"><button onclick="bleu();">fond bleu</button></div>
</body>
</html>

```

Deuxième exercice :

- Lorsqu'on appuie sur un bouton, on demande de rentrer un chiffre.



- Une fois le chiffre rentré, on l'affiche dans un paragraphe.



Remarque :

Pour récupérer la valeur du chiffre et l'afficher dans le paragraphe id= "text ", on utilisera la fonction : `document.getElementById("text").innerHTML=chiffre;`

## Correction:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>cours javascript1</title>
<style type="text/css">

        #bouton1 {
            position: fixed;
            top: 40px;
            left: 10px;
            z-index:10;
        }
</style>
</head>
<body>

<script type="text/javascript">
var chiffre;
function moar() {
    chiffre = prompt('Rentrer un chiffre:');
    document.getElementById("text").innerHTML=chiffre;
}
</script>
<div id="bouton1"><button onclick="moar()">Rentrer le chiffre</button></div>
<div id="main">
    <p>le chiffre est : <span id="text"></span></p>
</div>
</body>
</html>
```

### 5.1. La boucle while

Pour faire fonctionner une boucle, il est nécessaire de définir une condition. Tant que celle-ci est vraie (true), la boucle se répète. Dès que la condition est fausse (false), la boucle s'arrête.

Exemple : on va incrémenter un nombre, qui vaut 1, jusqu'à ce qu'il vaille 10 :

```
var number = 1;
while (number < 10) {
    number++;
}
alert(number); // Affiche : « 10 »
```

Au départ, number vaut 1. Arrive ensuite la boucle qui va demander si number est strictement plus petit que 10. Comme c'est vrai, la boucle est exécutée, et number est incrémenté. À chaque fois que les instructions présentes dans la boucle sont exécutées, la condition de la boucle est réévaluée pour savoir s'il faut réexécuter la boucle ou non. Dans cet exemple, la boucle se répète jusqu'à ce que number soit égal à 10. Si number vaut 10, la condition number < 10 est fausse, et la boucle s'arrête. Quand la boucle s'arrête, les instructions qui suivent la boucle (la fonction alert() dans notre exemple) sont exécutées normalement.

Pour arrêter une boucle en cours de fonctionnement, on peut utiliser la fonction **break**

## 5.2. La boucle do while

Elle ressemble très fortement à la boucle while, sauf que dans ce cas la boucle est toujours exécutée au moins une fois. Dans le cas d'une boucle while, si la condition n'est pas valide, la boucle n'est pas exécutée. Avec do while, la boucle est exécutée une première fois, puis la condition est testée pour savoir si la boucle doit continuer.

Exemple :

```
do {  
instruction_1;  
instruction_2;  
} while (condition);
```

## 5.3. La boucle for

La boucle for possède trois blocs qui la définissent. Le troisième est le bloc d'incrémentation qu'on va utiliser pour incrémenter une variable à chaque itération de la boucle.

Exemple :

```
for (var iter = 0; iter < 5; iter++) //(initialisation; condition; incrémentation)  
{  
alert('Itération n°' + iter);  
}
```

### Priorité d'exécution

Les trois blocs qui constituent la boucle for ne sont pas exécutés en même temps :

- *Initialisation* : juste avant que la boucle ne démarre. C'est comme si les instructions d'initialisation avaient été écrites juste avant la boucle, un peu comme pour une boucle while ;
- *Condition* : avant chaque passage de boucle, exactement comme la condition d'une boucle while;
- *Incrémentation* : après chaque passage de boucle. Cela veut dire que, si vous faites un break dans une boucle for, le passage dans la boucle lors du break ne sera pas comptabilisé.

## 6. Récupération des propriétés CSS

- La fonction `getComputedStyle()`

Il n'est pas possible de récupérer les valeurs des propriétés CSS d'un élément par le biais de la propriété `style` vu que celle-ci n'intègre pas les propriétés CSS des feuilles de style, ce qui nous limite énormément dans nos possibilités d'analyse.

La fonction `getComputedStyle()` va se charger de récupérer, à notre place, la valeur de n'importe quel style CSS ! Qu'il soit déclaré dans la propriété `style`, une feuille de style ou bien même encore calculé automatiquement, cela importe peu.

Exemple :

```
1<style type="text/css">  
2 #text {  
3 color: red;  
4 }  
5</style>  
6  
7<span id="text"></span>  
8  
9<script>  
10 var text = document.getElementById('text'),// document.getElementById retourne une référence à notre  
élément HTML text  
11 color = getComputedStyle(text, null).color;
```

```
12
13 alert(color);
14</script>
```

La solution `currentStyle` pour Internet Explorer 8 et antérieures

Pour Internet Explorer, il existe encore une petite différence car la fonction `getComputedStyle()` n'existe pas ! À la place nous allons donc nous servir de `currentStyle` :

Exercice :

- Réduire ou augmenter la dimension d'une image avec deux boutons.

Ajout de l'image :

```
<div align="center" id="cadre"></div>
```



Correction:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>cours javascript2</title>
<style type="text/css">

        #bouton1 {
            position: fixed;
            top: 10px;
            z-index:10;
        }
        #bouton2 {
            position: fixed;
            top: 30px;
            z-index:10;
        }

        #cadre {
            position: fixed;
            top: 200px;
            z-index:10;
        }

</style>
```

```

</head>
<body>

<script type="text/javascript">
var chiffre;
function agrandir() {
    var image = document.getElementById('image'),
        dimensionx = getComputedStyle(image, null).width;
    dimensiony = getComputedStyle(image, null).height;
    var dimensionx2=parseInt(dimensionx);
    dimensionx2=dimensionx2+40;
    var dimensiony2=parseInt(dimensiony);
    dimensiony2=dimensiony2+40;
    image.style.width=dimensionx2+"px";
    image.style.height=dimensiony2+"px";

}
function reduire() {
    var image = document.getElementById('image'),
        dimensionx = getComputedStyle(image, null).width;
    dimensiony = getComputedStyle(image, null).height;
    var dimensionx2=parseInt(dimensionx);
    dimensionx2=dimensionx2-40;
    var dimensiony2=parseInt(dimensiony);
    dimensiony2=dimensiony2-40;
    image.style.width=dimensionx2+"px";
    image.style.height=dimensiony2+"px";
}
</script>
<div align="center" id="bouton1"><button onclick="agrandir()">agrandir</button></div>
<div align="center" id="bouton2"><button onclick="reduire()">reduire</button></div>
<div align="center" id="cadre"></div>
</body>
</html>

```

- Cas particulier : Récupération de la valeur de la touche d'un clavier

Pour certain programme nous avons besoin de récupérer la valeur d'une touche de clavier (ex : jeux). On peut utiliser pour cela une fonction :

```
function toucheclavier(ev)
```

```

<body onkeyup="toucheclavier(event)">
<div id="out"></div>
</script>
</body>

```

`ev` étant une variable qui décrit l'événement qui s'est produit, en particulier dans le cas d'une touche pressée au clavier, elle contient l'information nous permettant de connaître cette touche. Le nom de la fonction n'est pas important.

Dans le cas d'un caractère imprimable, il suffit d'utiliser `ev.charCode` et la convention suivante :

32 : espace,  
36 : \$,  
60 : <,  
65 : A,

97 : a,  
etc.

Pour les autres touches, on utilise `ev.keyCode` et la convention suivante :

8 : backspace,  
37 : flèche gauche,  
38 : flèche haut,  
39 : flèche droite,  
40 : flèche bas.

Enfin, dans le cas d'un clic : `ev.pageX`, `ev.pageY`.

Exercice :

- On veut récupérer la variable d'une touche et l'afficher.

Correction :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>exercice touche</title>
</head>

<body>
<script type="text/javascript">
function toucheclavier(ev)
{
  var alt = ev.altKey;
  var shift = ev.shiftKey;
  var ctrl = ev.ctrlKey;
  var touche = ev.keyCode;
  var codeHTML="";

  if (shift)
    codeHTML += "shift + ";
  if (ctrl)
    codeHTML += "ctrl + ";
  if (alt)
    codeHTML += "alt + ";

  codeHTML += touche+"<br>";
  document.getElementById("out").innerHTML = codeHTML;
}
//-->
</script>

</head>

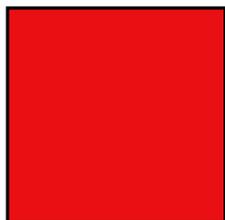
<body onkeyup="toucheclavier(event)">

<div >la valeur de la touche est : <span id="out"></span></div>
</script>
</body>
</html>
```

## 7. Animation 3D

### 7.1. Création de la page html et CSS

- Créer une première div dont l'id est cadre de dimension 200x200 avec un encadrement noir
- Puis à l'intérieur créer une deuxième div de même dimension dont l'id est boite avec un intérieur rouge
- Puis un bouton en dessous dont l'id est aller. Avec comme titre « tourner ». Puis un événement « onclick », qui lance la fonction « aller () » ;



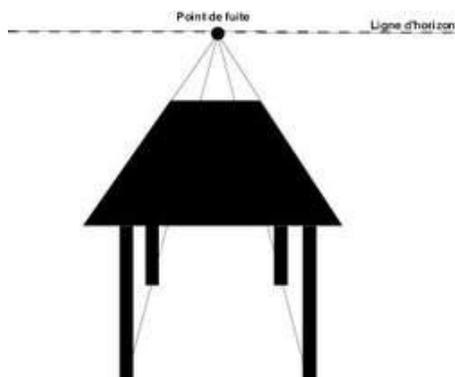
tourner

- Dans la page CSS :
  - Créer une animation « @keyframes spin1 », permettant une rotation sur deux axes (X et Y) sur 360 deg

```
@keyframes spin1 {  
    
}
```

Exemple : rotateY(30deg)

- Rajouter dans les paramètres du cadre la propriété « perspective » permettant de définir le point de fuite de la perspective



```
4 #cadre {  
5     
6     
7     
8   perspective:800px;  
9 }
```

- Rajouter dans les paramètres de la boite la propriété « transform-style » permettant de définir le type de transformation avec la caractéristique « preserve-3d »

```
#boite {  
    
  transform-style:preserve-3d;  
}
```

Valeurs :

### **preserve-3d**

Indique que les fils de l'élément ciblé doivent être positionnés dans l'espace tridimensionnel.

### **flat**

Indique que les fils de l'éléments sont positionnés dans le plan de l'élément.

- Rajouter l'élément suivant qui va lancer l'animation pendant 3s

```
#boite.next{
  animation:spin1 3s;
}
```

#### 7.2. Création de la page JavaScript

- Créer une fonction « aller(e) »
- Cette fonction va permettre d'ajouter la class « next » à la boite pendant la durée de l'animation puis l'enlever
- Rajouter le code suivant dans la fonction

```
boite.classList.add('next');
var temp=setTimeout(function(){boite.classList.remove('next');},3000);
```

Remarque :

#### **setTimeout indique un délai avant exécution**

Le méthode setTimeout définit une action à exécuter et un délai avant son exécution. Elle retourne un identifieur pour le processus.

Exemple : `var temps =setTimeout(mafonction, 5000);`

#### **ClearTimeout interrompt le décompte de setTimeout**

Cette méthode interrompt le délai et l'exécution du code associé à ce délai. Le processus à supprimer est reconnu par l'identifieur retourné par setTimeout.

Exemple : `var temps = setTimeout(mafonction,1000);`  
`clearTimeout(temps);`

#### **setInterval déclenche une opération à intervalles réguliers**

Similaire à setTimeout, elle déclenche répétitivement la même action à intervalles réguliers.

Exemple : `setInterval("alert('erreur')", 10000);`

- Tester l'application

<http://www.sti2dsinhyrome.fr/animation%203d/avec%20boutons/un%20bouton/index.html>

Code html :

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>animation 3d avec bouton</title>
<link href="indexcss.css" rel="stylesheet" type="text/css">
</head>

<body>
<script src="indexjs.js"></script>
<div id="cadre">
<div id="boite"></div>
</div>
<p>&nbsp;</p>
  <p>&nbsp;</p>
  <p>
    <input id="aller" type="button" value="tourner" onClick="aller();" />
  </p>

</body>
</html>
```

Code CSS :

```
1  @charset "utf-8";
2  /* CSS Document */
3
4  #cadre {
5      width: 200px;
6      height: 200px;
7      border: medium solid #000000;
8      perspective:800px;
9  }
10 #boite {
11     width: 200px;
12     height: 200px;
13     background-color: red;
14     transform-style: preserve-3d;
15 }
16 }
17 @keyframes spin1 {
18     from { transform: rotateY(0)
19         rotateX(0) ; }
20     to { transform: rotateY(360deg)
21         rotateX(360deg); }
22 }
23 #boite.next{
24     animation:spin1 3s;
25 }
```

Code JavaScript :

```
// JavaScript Document
function aller(e){
boite.classList.add('next');
var temp=setTimeout(function(){boite.classList.remove('next');},3000);
}
```