

Campus  
**technique**

8a, avenue V. Maistriau B-7000 Mons  
Tél : +32 (0)65 33 81 54  
Fax : +32 (0)65 31 30 51  
E-mail : tech-mons@heh.be

[www.heh.be](http://www.heh.be)

## UE : Développement front-end

- AA : Développement front-end - théorie  
[T-PINI-409]
- AA : Développement front-end – travaux pratique  
[T-PINI-410]

Bachelier en Informatique et systèmes  
orientation réseaux et télécommunications



# Index

<b>Index .....</b>	<b>1</b>
<b>Évaluation .....</b>	<b>2</b>
<b>Chapitre 1 : JavaScript.....</b>	<b>3</b>
1.1 Introduction au JavaScript.....	3
1.2 Intégration du JavaScript.....	6
1.3 JavaScript : langage de programmation objet.....	9
1.4 JavaScript : langage de programmation événementiel .....	12
Récapitulatif des événements JavaScript .....	13
1.5 Variables .....	14
1.6 Tableaux .....	17
1.7 Opérateurs.....	19
Exercice 1 : tests sur les variables et les opérateurs .....	20
1.8 Structures conditionnelles .....	21
1.9 Structures itératives.....	23
1.10 Fonctions .....	25
Exercice 2 : imposer la majuscule dans un champ texte .....	27
Exercice 3 : changer la source d'une image lors d'un clic.....	28
Exercice 4 : le Morpion .....	29
1.11 DOM .....	30
Exercice 5 : Total = nombre * prix à l'unité.....	32
Exercice 6 : le Formulaire Dynamique .....	33
Exercice 7 : les Flèches.....	34
<b>Chapitre 2 : jQuery .....</b>	<b>35</b>
2.1 Introduction.....	35
2.2 Mise en place de jQuery.....	36
2.3 Utilisation de jQuery .....	36
2.4 Les événements jQuery.....	38
2.5 Les animations jQuery.....	44
2.6 Ordre des animations.....	46
2.7 Modification d'attributs.....	47
2.8 Exemple d'utilisation d'AJAX avec jQuery.....	49
Exercice 8 : les Véhicules.....	50
Exercice 9 : la Bulle .....	51
Exercice 10 : Consonnes et Voyelles.....	52
Exercice 5 : la bulle .....	53
Examen de labo juin 2013 : la course d'oryctéropes.....	54
Examen de labo juin 2013 : Square-memory .....	55
<b>Chapitre 3 : CSS3 .....</b>	<b>56</b>
3.1 Compatibilité de CSS3 .....	56
3.2 CSS3 comme alternative à jQuery .....	57
A. Transitions .....	57
B. Animations et <i>keyframes</i> .....	57
C. Comment choisir entre CSS3 et jQuery ? .....	58
<b>Chapitre 4 : responsive design .....</b>	<b>59</b>
4.1 Introduction.....	59
4.2 Site dédié au mobile, application mobile ou site responsive ?.....	59

4.3 Statique, fluide, adaptatif ou <i>responsive</i> ? .....	61
4.4 En pratique .....	62
4.5 FlexBox .....	63
4.6 Plus d'information sur ce sujet ? .....	65
<b>Récapitulatifs</b> .....	<b>66</b>
Récapitulatif HTML .....	66
Récapitulatif des sélecteurs CSS .....	68
Récapitulatif des propriétés CSS .....	69
<b>Projet : Portfolio OnePage Responsive</b> .....	<b>72</b>
<b>Sources</b> .....	<b>74</b>

## Évaluation

- **Interro écrite (20%)** : non remédiable en 2<sup>e</sup> session et dont la matière porte sur l'ensemble de ce syllabus
- **Projet (20%)** : non remédiable en 2<sup>e</sup> session et dont les consignes se trouvent à la fin de ce syllabus
- **Examen de labo (60%)** : durée 2h, a lieu sur les machines de l'école, sans internet et sans supports électroniques (clé USB, calculatrice, GSM, disque dur, etc.), mais à cours papier ouvert (syllabus, notes personnelles, codes sources imprimés, livres, etc.)

Titulaire : Ivan Miller

Contact : [ivan.miller@heh.be](mailto:ivan.miller@heh.be)

# Chapitre 1 : JavaScript

## 1.1 Introduction au JavaScript

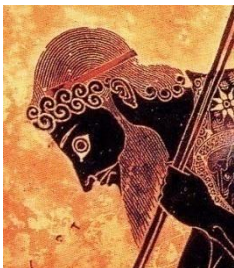
### A. Historique

**JavaScript est un langage Web dynamique, qui s'intègre aux pages Web afin d'améliorer leur interactivité.**

JavaScript était au départ un projet de langage Web côté serveur, nommé LiveScript et développé par Netscape. Adopté à la fin de l'année 1995 par la firme Sun (qui a aussi développé Java), il change alors son nom en JavaScript.

A l'époque, le JavaScript souffrait de nombreuses incompatibilités entre navigateurs, ce qui limitait son utilisation. De plus, la lenteur des connexions internet dissuadait les développeurs Web à utiliser pleinement ce langage. C'est pourquoi le succès de JavaScript a eu des débuts difficiles.

Pourtant, JavaScript a connu un regain d'intérêt peu après l'an 2000 grâce à trois facteurs : le premier est la sortie d'**Internet Explorer 3** en 1998 qui supportait enfin le langage, le second est l'arrivée de la connexion à haut débit (**ADSL**) qui met un terme aux anciennes connexions modem, et donc aux lenteurs de chargement de certains scripts conséquents et le dernier est l'arrivée d'**Ajax**.



**Ajax** fut d'abord implémenté par Microsoft et ensuite généralisé aux autres navigateurs. L'objet *XMLHttpRequest* (appelé aussi *XHR*) permet la communication en arrière-plan entre le navigateur et le serveur, permettant à un script de charger des données sans recharger complètement une page. Cette technologie s'appelle : *Asynchronous JavaScript and XML* ou tout simplement Ajax.

Un bel exemple d'implémentation d'Ajax est le site Google Maps, où vous pouvez vous déplacer dans une carte pendant que des scripts en arrière-plan, téléchargent des portions de cartes. L'utilisation d'AJAX se justifie le plus souvent par un confort d'utilisation (pas de rechargement de la page), ou par une nécessité technique (trop de contenu à charger ou contenu à mettre à jour en *live*).

N'oublions pas que JavaScript a, au début, été intégré aux navigateurs pour de petits scripts destinés à de simples tâches côté client et qu'il est toujours utilisé de cette façon sur le Web.

**Les atouts de JavaScript** sont donc de permettre :

1. Les traitements (variables, calculs, fonctions, etc.)
2. La détection d'évènements (interactions de l'utilisateur avec l'interface)
3. L'utilisation d'Ajax
4. L'exploitation du DOM

## B. JavaScript n'est pas Java

JavaScript	Java
<ul style="list-style-type: none"><li>• Code intégré dans la page HTML</li><li>• Code interprété par le navigateur</li><li>• Scripts simples pour des applications limitées</li><li>• Permet d'accéder aux objets du navigateur</li><li>• Confidentialité des codes nulle (code source lisible par le visiteur)</li></ul>	<ul style="list-style-type: none"><li>• Module (applet) distinct de la page HTML</li><li>• Code source compilé avant son exécution</li><li>• Langage de programmation complexe et plus performant</li><li>• N'accède pas aux objets du navigateur</li><li>• Code sécurisé puisque « broyé » lors de la compilation</li></ul>

## C. Les outils nécessaires

Pour apprendre et utiliser le JavaScript, il vous faut :

- Un **navigateur** qui reconnaît le JavaScript (n'importe quel navigateur récent) avec outils de développement intégrés
- Un simple éditeur de texte (par exemple **Notepad++**)
- De bonnes connaissances en **HTML** et en **CSS**

Comme le JavaScript vient s'ajouter au code HTML, une connaissance approfondie des éléments HTML est indispensable. C'est pourquoi, les utilisateurs d'éditeurs HTML de type *wysiwyg* peuvent retourner au vestiaire. Car si les éditeurs HTML tels que DreamWeaver, Claris Home Page et d'autres facilitent la création de documents HTML, lorsqu'il s'agira de JavaScript il vous faudra retrousser vos manches et attaquer le code source.

## D. Questions sur les langages de script

### Quelle différence entre un langage de script et un langage de programmation ?

JavaScript est un langage de script. Le langage C est un langage de programmation. Heureusement, nous ne sommes pas tenus à tant de rigueur dans le choix de nos termes et rien ne nous empêche d'annoncer dans notre CV que nous maîtrisons le langage de programmation JavaScript.

Certains langages de programmation sont **compilés**, c'est-à-dire traduits en langage machine avant leur exécution. JavaScript pour sa part, comme le HTML, est **interprété** : le navigateur exécute chaque ligne du programme au fur et à mesure.

### Quel est l'avantage des langages de programmation interprétés ?

Le principal avantage est que l'écriture ou la modification d'un script est plus rapide. Le changement prend automatiquement effet dès que l'on recharge le document dans le navigateur.

De plus, ils sont sensés être davantage portables que les langages compilés, puisque ces derniers nécessitent l'installation de certains programmes.

### Et les désavantages ?

Les langages interprétés seront toujours moins rapides que les langages compilés. N'oublions pas non plus, qu'ils nécessitent un interpréteur.

## 1.2 Intégration du JavaScript

### A. Comment l'intégrer ?

L'élément HTML `<script>` signale au navigateur les portions de code en JavaScript :

```
<script>... </script>
```

**Remarque** : l'attribut `type="text/javascript"` était obligatoire avant l'HTML5.

La balise `<script>` peut être insérée n'importe où dans la page HTML. Cependant, il ne faut pas perdre de vue que le navigateur traite le document HTML de haut en bas, y compris le JavaScript. Par conséquent, une instruction ne pourra être exécutée convenablement que si le navigateur possède au préalable tous les éléments nécessaires à son exécution.

Certains scripts ont leur place à la fin du `<head>` afin d'être chargés en premier lieu alors que d'autres, moins importants ou plus lents, ont leur place à la fin du `<body>` pour ne pas retarder l'affichage de la page. C'est par exemple le cas du script de Google Analytics qui, malgré sa rapidité, n'apporte rien à la page et aura donc sa place à la fin du `<body>`.

De manière générale, la plupart des scripts se lancent après le chargement des éléments de la page. Il est donc préférable de les placer juste avant `</body>` de façon à ne pas retarder inutilement l'affichage du contenu de la page.

### B. Scripts externes

Il est conseillé d'utiliser des fichiers externes pour plus de lisibilité. Ainsi, on peut stocker les scripts dans des fichiers ".js" et les appeler à partir d'un fichier HTML. Cela permet de constituer une bibliothèque de scripts aisément réutilisables.

```
<script src="monscript.js"></script>
```

**Remarque** : si l'élément `<script>` comporte une source externe (`src`), il ne doit pas comporter de code entre son ouverture et sa fermeture.

## C. Masquer les scripts pour les anciens navigateurs

Les navigateurs qui ne comprennent pas le JavaScript ignorent la balise `<script>` et affichent le code du script comme s'il s'agissait d'un simple texte. Pour éviter ce problème d'affichage, on utilisera les balises de commentaires du langage HTML :

```
<script>
<!--
...
-->
</script>
```

Le comportement des navigateurs face à une instruction inconnue est de passer cette instruction. Les navigateurs ne comprenant pas la balise `<script>` vont donc passer au commentaire comme s'il s'agissait de code HTML et n'afficheront pas le code dans la page.

Cependant, le JavaScript est supporté par presque tous les navigateurs. Seules les versions 1 et 2 d'Internet Explorer et la version 1 de Netscape poseront problème. Autant dire que nous sommes à l'abri.

## D. Les commentaires

JavaScript utilise les mêmes conventions que le C et le C++ pour les commentaires :

```
<script>
//commentaire jusqu'en fin de ligne

/*commentaire
sur plusieurs
lignes*/
</script>
```

## E. Utilisation du point-virgule

En JavaScript, contrairement au langage C ou au PHP, le point-virgule est un séparateur d'instruction et pas un terminateur d'instruction. Cette particularité permet de ne pas coder le point-virgule d'une instruction qui ne serait pas suivie d'une autre instruction.



## F. Messages console

Afin de faciliter le débogage, JavaScript permet d'afficher des messages dans la console : par exemple du texte ou des valeurs de variables.

```
console.log(...) ;
```

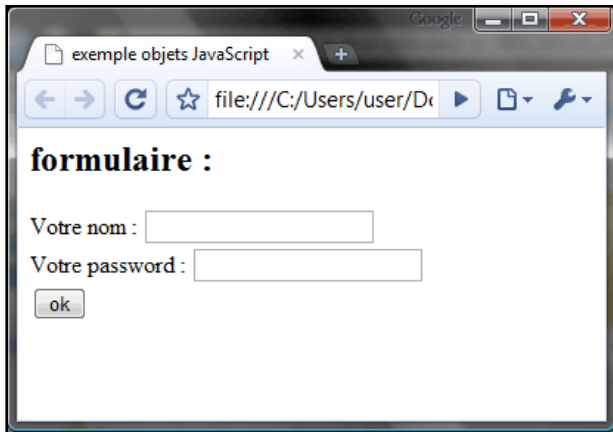
Pour une utilisation plus pointue de la console, n'hésitez pas à vous renseigner sur les fonctions suivantes :

```
console.table() pour un affichage complet d'un tableau ;  
console.time() et console.timeEnd() pour chronométrer les scripts ;  
console.group() ;  
console.dir() ;  
console.assert() ;  
console.trace() ;  
console.info() ;  
console.warn() ;  
console.error() ;
```

## 1.3 JavaScript : langage de programmation objet

### A. Les objets et leurs méthodes

**JavaScript est un langage orienté objet.** La page web affichée à l'écran est considérée par le JavaScript comme un ensemble d'objets imbriqués que nous allons pouvoir manipuler.



La page web ci-contre est divisée en plusieurs objets imbriqués :

- L'objet fenêtre : **window**
- L'objet document : **document**
- L'objet formulaire : **form**

L'objet formulaire contient à son tour un objet texte, un objet password et un objet bouton.

A chaque objet JavaScript est dédié un certain nombre de **méthodes** (= fonctions propres à l'objet) et de **propriétés** (= variables propres à l'objet). L'utilisation de ces méthodes et de ces propriétés se fait de la façon suivante :

```
nom_de_l_objet.nom_de_la_methode()  
nom_de_l_objet.nom_de_la_propriete
```

Illustrons ceci par l'exemple d'une première méthode : la méthode *write()* propre au document :

```
document.write();
```

Comme le premier objet de la page est l'objet **window**, il est possible d'utiliser ses méthodes sans préciser le nom de cet objet. Ainsi, les deux écritures suivantes ont le même effet :

```
window.alert("Bonjour");  
alert("Bonjour");
```

### B. Les méthodes *alert*, *confirm* et *prompt*

Voici 3 méthodes de l'objet *window* qui permettent d'afficher une petite fenêtre d'alerte :

*window.alert()* affiche une fenêtre avec un texte d'alerte et un bouton "OK".  
*window.confirm()* affiche une fenêtre avec un texte et deux boutons : "OK" et "Annuler".  
*window.prompt()* affiche une fenêtre avec un champ texte à compléter.

## C. Créer un objet

Avec la syntaxe suivante, vous pouvez facilement créer vos propres objets :

```
var monBateau = new Object();
monBateau.nom = "le Ribouflard";
monBateau.longueur = 83.6;
monBateau.equipage = 46;
```

Autre syntaxe :

```
var maVoiture = {
  marque: "Toyota",
  modele: "Yaris",
  annee: 1998
};
```

## D. L'objet *Math* et ses méthodes

En langage C, on utilise une bibliothèque nommée *math.h* contenant toutes les fonctions mathématiques. En JavaScript, **Math** est un objet (et pas une bibliothèque) et joue plus ou moins le même rôle.

Méthodes :

```
Math.ceil()      //borne supérieure
Math.floor()     //borne inférieure
Math.round()     //arrondi au plus proche
Math.random()    //génère un nombre aléatoire entre 0 et 1
Math.sqrt()      //racine carrée
Math.cos()       //cosinus
Math.min(4,2,8)  //minimum (dans ce cas, vaut 2)
Math.max(4,2,8)  //maximum (dans ce cas, vaut 8)
...
```

Propriétés constantes :

```
Math.PI          //constante pi
Math.E           //constante e
...
```

## E. *null*

Le mot *null* représente un objet vide, sans valeur.

## F. L'objet *this*

L'objet *this*, représente l'objet dans lequel on se trouve. Par exemple lors d'un appel de fonction à partir d'un élément HTML, *this* va représenter cet élément.

Il s'agit d'un raccourci d'écriture bien pratique. Voici un exemple où *this* représente le `<div>`.

```
<div onClick="clic_div(this)">
```

Dans cet exemple, la fonction `clic_div()` va recevoir l'objet `<div>` afin de pouvoir par exemple en modifier les propriétés ou le contenu.

## G. Propriétés HTML et CSS

Chaque objet HTML dispose d'autant de propriétés qu'il possède d'attributs. Il est donc possible en JavaScript de modifier les valeurs de ces attributs.

Par exemple un objet `<img>` appelé `x` disposera des propriétés suivantes :

- `x.src`
- `x.alt`
- `x.width`
- ...

Ce même objet disposera également de l'objet `style` si des **styles CSS** lui sont appliqués. Cet objet `style` dispose de toutes les propriétés CSS possibles, mais écrites en **camelCase** :

- `x.style.border`
- `x.style.left`
- `x.style.backgroundColor`
- `x.style.marginTop`
- ...

## 1.4 JavaScript : langage de programmation évènementiel

### A. Les évènements

**Les évènements sont les actions de l'utilisateur** : cliquer, survoler, redimensionner la fenêtre, charger la page, défiler la page, etc.

En HTML, il n'existe (presque) qu'un seul évènement : le clic de la souris. Heureusement et pour notre plus grand plaisir, JavaScript propose une dizaine d'évènements supplémentaires qui rendront nos pages plus interactives.

JavaScript permet d'associer ces évènements à des actions (fonctions).

### B. Exemples

#### Afficher des messages au chargement et à la fermeture d'une page Web

```
<body onload="alert('Bienvenue')" onunload="alert('bye bye')"></body>
```

#### Afficher le nombre de clics sur un élément

```
<h1 onclick="clic()">cliquez-moi</h1>

<script>
nb=0;
function clic(x) {
    nb++;
    alert(nb);
}
</script>
```

#### Changer la source d'une image avec *onMouseOver* et *onMouseOut*

```


<script>
function curseurOver(x) {
    x.src="tyrannosaurus.png";
}
function curseurOut(x) {
    x.src="ankylosaure.png";
}
</script>
```

## Récapitulatif des évènements JavaScript

Evènements	Descriptions
onClick	Clic
onDbClick	Double clic
onChange	Changement de la valeur d'un champ de formulaire
onMouseOver	Survol du curseur de la souris
onMouseOut	Fin du survol du curseur de la souris
onMouseDown	Bouton de la souris enfoncé
onMouseMove	Survol et mouvement du curseur de la souris
onMouseUp	Bouton de la souris relâché
onMove	Mouvement
onHelp	Aide (F1)
onSelect	Sélection d'un élément
onSubmit	Envoi d'un formulaire
onResize	Redimensionnement de la page
onReset	Suppression du contenu d'un formulaire
onFocus	Sélection d'un champ du formulaire
onKeyPress	Touche du clavier pressée
onBlur	Perte du focus
onScroll	Utilisation des barres de défilement
onLoad	Chargement (de la page par exemple)
onUnload	Fermeture de la page
onError	Erreur
onAbort	Annulation

## 1.5 Variables

### A. Noms de variables

Comme toujours, les variables sont des conteneurs de données. Le processeur les repère par leur adresse, alors que le programmeur les repère par leur nom (qu'il doit choisir).

Règles sur les **noms de variables** :

- Le nom doit commencer par une lettre, un "\_" ou un "\$" ;
- Le nom doit se composer uniquement de lettres (y compris lettres accentuées), de chiffres et des caractères "\_" et "\$" ;
- Les caractères accentués ne peuvent pas être suivis d'un chiffre ;
- Le JavaScript est sensible à la casse (attention aux majuscules) ;
- Le nom ne peut pas être un nom réservé.

Les mots ci-dessous sont des mots clefs du JavaScript et sont donc réservés :

<b>abstract</b>	<b>boolean</b>	<b>break</b>	<b>byte</b>	<b>case</b>
<b>catch</b>	<b>char</b>	<b>class</b>	<b>const</b>	<b>continue</b>
<b>default</b>	<b>do</b>	<b>double</b>	<b>else</b>	<b>extends</b>
<b>false</b>	<b>final</b>	<b>finally</b>	<b>float</b>	<b>for</b>
<b>function</b>	<b>goto</b>	<b>if</b>	<b>implements</b>	<b>import</b>
<b>in</b>	<b>instanceof</b>	<b>int</b>	<b>interface</b>	<b>long</b>
<b>native</b>	<b>new</b>	<b>null</b>	<b>package</b>	<b>private</b>
<b>protected</b>	<b>public</b>	<b>return</b>	<b>short</b>	<b>static</b>
<b>super</b>	<b>switch</b>	<b>synchronized</b>	<b>this</b>	<b>throw</b>
<b>throws</b>	<b>transient</b>	<b>true</b>	<b>try</b>	<b>var</b>
<b>void</b>	<b>while</b>	<b>with</b>		

Conseils : choisissez toujours des noms de variables représentatifs de leur contenu.

Choisissez une écriture commune à toutes vos variables :

- `ageClient` : *camelCase* ou écriture en dos de chameau
- `age_client` : écriture avec underscore à la place des espaces

### B. Déclarations explicites et implicites

Les variables peuvent se déclarer de 2 façons :

- **Déclaration explicite** : on annonce au JavaScript de nouvelles variables. Exemples :  

```
var age = 45 ;
```

```
var prenom = "Sophie" ;
```
- **Déclaration implicite** : on affecte une variable non déclarée et JavaScript s'en accommode. Exemples :  

```
age = 45 ;
```

```
prenom = "Sophie" ;
```

Pour votre facilité, il est tout de même conseillé d'utiliser la façon explicite.

## C. Types de données

JavaScript utilise seulement quatre types de données :

- les **nombres** : entiers ou avec virgule tels que 45 ou 3.1416 ; le nombre peut commencer par 0 pour une écriture octale ou par 0x pour une écriture hexadécimale.
- les **chaînes de caractères** : c'est-à-dire des suites de caractères entre guillemets telles que "Isabelle"
- les **booléens** : qui valent *true* pour vrai et *false* pour faux
- les **undefined** : les variables qui n'ont pas de type. Exemple :  
`var bidule ;`

Notons que contrairement à d'autres langages, en JavaScript, on ne précise pas le type de données lors de la déclaration. Pourquoi ? Parce que **JavaScript est un langage non typé !**

Pour faire simple, dans les langages non typés, le type des variables existe mais n'est pas figé. Il est donc possible d'affecter un nombre à une variable, puis d'y affecter une chaîne de caractères ou un booléen. Exemple :

```
x = 8 ;  
x = "Barnabé" ;
```

Comment connaître le **type** d'une variable ?

```
alert( typeof(x) ) ;
```

Comme le type n'est pas figé mais qu'il existe bel et bien, toutes les erreurs de programmation liées aux types sont possibles. Chouette...

**Exemple d'erreur :**

```
ab = "gladi";  
cd = "ateur";  
alert(ab*cd);
```

Multiplier des chaînes de caractères est une erreur. C'est pourquoi, ce script va afficher l'expression "**NaN**" signifiant "**Not a Number**".

**Autre exemple d'erreur :**

```
z=8 ;  
alert(z.length) ;
```

**length** est une propriété des chaînes de caractères. Ce script va afficher « **undefined** ».



## D. Conversions de types

Les erreurs de programmation dues aux types des variables sont fréquentes. Heureusement, nous avons à disposition une série de fonction permettant de convertir les données.

<code>parseInt(x)</code>	Convertit x en nombre entier
<code>parseInt(x, b)</code>	Convertit x en nombre entier avec x écrit en base b
<code>parseFloat(x)</code>	Convertit x en nombre réel
<code>String(x)</code>	Convertit x en chaîne de caractères

Autres conversions possibles :

<code>x.toString()</code>	Méthode convertissant x en chaîne de caractères
<code>x=y*1</code>	Multiplier la chaîne de caractères y par 1 fera de x un nombre

## E. Méthodes et propriétés des objets String

<code>a.indexOf('x')</code>	Donne la position de la première occurrence de la lettre 'x' dans la chaîne <i>a</i>
<code>a.lastIndexOf('x')</code>	Donne la position de la dernière occurrence de la lettre 'x' dans la chaîne <i>a</i>
<code>a.charAt(8)</code>	Donne le 9 <sup>e</sup> caractère de la chaîne <i>a</i>
<code>a.length</code>	Contient le nombre de caractères de la chaîne <i>a</i>
<code>a.substring(4,10)</code>	Donne les caractères de la chaîne <i>a</i> compris entre le 5 <sup>e</sup> et le 11 <sup>e</sup>
<code>a.substring(4)</code>	Donne les caractères de la chaîne <i>a</i> depuis le 5 <sup>e</sup> jusqu'à la fin
<code>a.substr(4,10)</code>	Donne les caractères de la chaîne <i>a</i> compris entre le 5 <sup>e</sup> et le 15 <sup>e</sup>
<code>a.toLowerCase()</code>	Donne la chaîne réécrite en minuscules
<code>a.toUpperCase()</code>	Donne la chaîne réécrite en majuscules

## F. Portée des variables globales et locales

La portée d'une variable représente les endroits du code où l'on peut utiliser cette variable. La portée dépend de la déclaration (implicite ou explicite) et de son emplacement.

Les variables déclarées en dehors des fonctions, sont toujours **globales**, qu'elles soient déclarées explicitement ou implicitement. On peut les utiliser partout dans le script.

Les variables déclarées explicitement dans une fonction, ont une portée limitée à cette seule fonction. Elles sont dites **locales**.

Par contre, les variables déclarées implicitement dans une fonction sont **globales**.

**Toutefois, les variables globales sont à éviter au maximum pour cause de surcharge inutile et de risques de répétitions de noms de variables.**

## 1.6 Tableaux

### A. Tableaux indicés

En JavaScript, il existe 2 sortes de tableaux : les tableaux indicés et les tableaux associatifs. Dans le cas des tableaux indicés, les indices sont des nombres entiers. Dans le cas des tableaux associatifs, les indices sont des chaînes de caractères.

**Déclaration d'un tableau :**

```
var semaine = new Array(7);
```

**Déclaration avec initialisation :**

```
var semaine = new Array("lundi","mardi",...,"dimanche") ;  
var points = new Array(15,7,9,10,2,18,14,12,10,9,9) ;
```

**Affectations :**

```
semaine[0]="lundi";  
semaine[1]="mardi";  
...
```

**Affichage :**

```
for (i=0 ; i<semaine.length ; i++) {  
    document.write(semaine[i]+"<br>") ;  
}
```

### B. Tableaux associatifs

**Déclaration :**

```
var navig = { "un" : 1, "deux" : 2, "trois": 3};
```

**Exemple d'affectations :**

```
navig["un"]=...;  
navig["deux"]=...;  
navig["trois"]=...;
```

**Remarque :** il sera plus aisé de parcourir les tableaux associatifs avec une boucle *for ( in )*.

Les tableaux associatifs peuvent donc être utilisés comme des tableaux indicés afin de récupérer ou d'utiliser les indices.

## G. Tableaux indicés à plusieurs dimensions

### Déclaration et initialisation d'un tableau de 2 lignes et 2 colonnes :

```
var tableau = new Array(); //déclaration d'un tableau à 1 dimension
tableau[0] = new Array(); //hop, le 1e élément devient un tableau
tableau[1] = new Array(); //hop, le 2e élément devient un tableau

tableau[0][0] = "Julien"; //exemples d'affectations
tableau[0][1] = 32; //on peut stocker des types différents
tableau[1][0] = "Quentin";
tableau[1][1] = 26;
```

### Variante où l'on colle 2 tableaux à 1 dimension en un tableau à 2 dimensions :

```
aNoms = new Array("Nom1", "Nom2", "Nom3");
aPrenom = new Array("Prénom1", "Prénom2", "Prénom3");
monTableau = new Array(aNoms, aPrenom);
```

### Variante avec initialisation :

```
tab = new Array(["Bob", "Luc", "Joe"], ["Bill", "Luca", "Jim"],
["Ben", "Guy", "Léon"]);
```

## H. Quelques méthodes et propriétés de l'objet Array

Soit *tableau* le nom d'un objet *Array* :

*tableau.length* : nombre d'éléments du tableau.

*tableau.concat()* : regroupe plusieurs tableaux en un seul.

*tableau.join()* : regroupe tous les éléments du tableau en une seule chaîne de caractères.

*tableau.reverse()* : inverse l'ordre des éléments.

*tableau.sort()* : trie les éléments d'un tableau par ordre alphabétique.

## 1.7 Opérateurs

### Opérateurs arithmétiques :

Addition	+	
Soustraction	-	
Multiplication	*	
Division Entière	/	Uniquement sur des types entiers
Modulo	%	Reste de la division entière
Division Réelle	/	Uniquement sur des types réels
Affectation	=	

### Opérateurs d'incrémentation :

Incrémentation	++i ;	ou	i++ ;
Décrémentation	--i ;	ou	i-- ;

### Opérateurs d'affectation élargie (raccourci d'une affectation et d'une opération) :

i += k ;	équivalent à	i = i + k ;
i -= k ;	équivalent à	i = i - k ;
i *= k ;	équivalent à	i = i * k ;
i /= k ;	équivalent à	i = i / k ;
i %= k ;	équivalent à	i = i % k ;

### Opérateur de concaténation :

Concaténation	+	Uniquement si l'une des valeurs est une chaîne de caractères
---------------	---	--

### Opérateurs logiques :

AND	&&
OR	
Contraire	!

### Opérateurs relationnels (pour les comparaisons) :

Plus grand que	>
Plus petit que	<
Plus grand ou égal à	>=
Plus petit ou égal à	<=
Égal à	= =
Égal à (type et valeur)	= = =
Différent de	!=

**Remarque : ne confondez pas = et ==. L'un est une affectation (changement de la valeur), l'autre une comparaison (souvent utilisée dans les conditions).**

## Exercice 1 : tests sur les variables et les opérateurs

### Principe :

Dans cet exercice, nous allons mettre en pratique les notions de variables, de types et de concaténations en pratiquant 4 tests dont les résultats seront affichés avec la méthode `alert()`.

### Démarche :

Créez une page HTML.

Insérez un élément `<script>` avant la fermeture `</body>`.

### Test 1 :

Déclarez explicitement et initialisez les variables `a1` à 12 et `a2` à 8.

Affichez `a1+a2` via la méthode `alert()`.

**Question** : à quelle opération correspond ici l'opérateur `+` ?

### Test 2 :

Affectez maintenant la valeur `"Raoul"` à la variable `a1`.

Affichez `a1+a2` via la méthode `alert()`. Que constatez-vous ?

**Question** : à quelle opération correspond ici l'opérateur `+` ?

### Test 3 :

Toujours avec la méthode `alert()`, affichez maintenant `"a1+a2"` (n'oubliez pas les guillemets).

**Question** : quel rôle joue ici le `+` ?

### Test 4 :

Toujours avec la méthode `alert()`, affichez maintenant `'a1+a2'` (n'oubliez pas les apostrophes).

**Question** : remarquez-vous une différence avec le résultat du test 3 ?

## 1.8 Structures conditionnelles

Les structures conditionnelles sont des tests permettant d'exécuter ou non une série d'instructions.

### A. *if*

L'instruction *if* est la structure de test la plus basique, on la retrouve dans tous les langages de programmation (avec une syntaxe parfois légèrement différente). Elle permet de poser une condition au programme pour que certaines instructions soient réalisées.

```
if (condition) instruction1 ;  
else instruction2 ;
```

Une condition n'a que deux résultats possibles : vrai ou faux.

Si la condition est vraie, l'instruction 1 est exécutée, si la condition est fausse, l'instruction 2 est exécutée.

#### Remarques :

- La condition est toujours entre parenthèses.
- Comme les instructions après le *else* ne sont exécutées que si la condition du *if* est fausse, il est possible de poser une seconde condition seulement si la première était fausse, comme suit :

```
if (condition1) instruction1 ;  
else if (condition2) instruction2 ;  
else instruction3 ;
```

- Il est possible de poser des conditions multiples liées par les opérateurs logiques AND (&&) ou OR (||).

```
if ((condition1)&&(condition2)) instruction ;
```

- Si les instructions sont multiples, il faut les mettre entre accolades :

```
if (condition) {  
    instruction1 ;  
    instruction2;  
}
```

## B. *switch*

L'instruction *switch* permet de faire plusieurs tests de valeurs sur le contenu d'une même variable.

Exemple de syntaxe :

```
switch (variable) {  
    case 1 :    instructions1;  
                break;  
    case 2 :    instructions2;  
                break;  
    case 3 :    instructions3;  
                break;  
    default : instructions_par_defaut;  
}
```

Les parenthèses qui suivent le mot clé ***switch*** indiquent une variable dont la valeur est testée successivement par chacun des ***case***. Lorsque la variable testée est égale à une des valeurs suivant un *case*, la liste d'instructions qui suit est exécutée.

Le mot clé ***break*** indique la sortie de la structure conditionnelle.

Le mot clé ***default*** précède les instructions qui seront exécutées si l'expression n'est égale à aucune des valeurs proposées dans les *case*.

## C. Opérateur ternaire : ?

L'opérateur ternaire peut dans certains cas remplacer une structure *if else*. Ses possibilités sont plus limitées, mais son exécution est plus rapide.

```
(age>=18) ? acces=TRUE : acces=FALSE ;
```

Autre exemple d'utilisation :

```
acces = (age>=18) ? TRUE : FALSE ;
```

## 1.9 Structures itératives

Les structures itératives, également appelées boucles, permettent de répéter des instructions.

### A. *while*

```
while (condition) {  
    instructions;  
}
```

Cette instruction exécute les instructions **tant que** (*while* est un mot anglais qui signifie *tant que*) la condition est vraie.

Le risque est que la condition reste vraie... toujours ! On appelle cela une **boucle infinie**, c'est-à-dire un plantage du programme qui n'arrive jamais à sortir de la boucle en question.

### B. *do while*

L'instruction *do while* est semblable à l'instruction *while* à ceci près que la condition se trouve en fin de boucle. Le test de la condition ayant lieu après les instructions, celles-ci sont exécutées au moins une fois par le programme (même si la condition est fausse).

Cette boucle convient lorsque la condition porte sur un élément n'étant pas initialisé avant la boucle (comme l'élément n'a pas de valeur, il est impossible de l'utiliser).

La syntaxe est la suivante :

```
do {  
    instructions;  
} while (condition);
```

### C. *for*

La troisième structure itérative est l'instruction *for*. Comme pour l'instruction *while*, la condition se fait avant les instructions, mais cette fois avec une écriture condensée comprenant l'affectation du compteur, la condition et la progression (incrémentation par exemple) du compteur.

La syntaxe est la suivante :

```
for (affectation ; condition ; progression) {  
    instructions;  
}
```



## D. *for in*

Petite variante du *for*, le *for in* permet d'exploiter toutes les propriétés d'un objet (voir toutes les valeurs d'un tableau, puisque les tableaux sont des objets).

La syntaxe est la suivante :

```
for (var prop in mon_objet) {  
    document.write("nom : "+prop+"/ valeur : "+mon_objet[prop]) ;  
}
```

## E. *break* et *continue*

L'instruction *break* permet d'interrompre prématurément une boucle ou une structure conditionnelle.

L'instruction *continue* permet de sauter une instruction dans une boucle, tout en la continuant.

## F. *with*

Vous pouvez utiliser le mot clé *with* pour effectuer plusieurs instructions consécutives sur une même variable. Cela permet un gain de temps lorsqu'il s'agit d'une longue série d'opérations.

Exemple :

```
with (nom) {  
    window.alert("Longueur du nom : " + length);  
    window.alert("Le nom est : " + toUpperCase());  
}
```

Au lieu de :

```
window.alert("Longueur du nom : " + nom.length);  
window.alert("Le nom est : " + nom.toUpperCase());
```

Ce n'est que lorsque le nombre d'instructions est considérable que cela devient vraiment intéressant.

## 1.10 Fonctions

Une fonction contient un extrait de code susceptible d'être réutilisé. Pour ne pas coder plusieurs fois cet extrait de code, on l'isole dans une fonction que l'on appelle au besoin.

### A. Déclaration des fonctions

La déclaration d'une fonction se fait par l'utilisation du mot clé *function*. Exemple :

```
function nom_de_la_fonction (paramètres) {  
    //code de la fonction  
}
```

Le nom de la fonction doit respecter **les mêmes règles** que pour les noms de variables.

La présence de **paramètres** est facultative, mais les parenthèses sont obligatoires. C'est d'ailleurs grâce à ces parenthèses que l'interpréteur JavaScript distingue les fonctions des variables.

Il est judicieux de placer toutes les déclarations de fonctions dans le *<head>*, ou mieux encore, de les placer dans un fichier JavaScript externe importé dans le *<head>* et ce pour 2 raisons :

1. Comme l'interpréteur lit le code de haut en bas, il faudra veiller à ce que la fonction soit déclarée avant d'être appelée (sauf si l'appel est fait à partir d'un événement).
2. Le fait de déclarer une fonction n'entraîne pas son exécution, donc la présence de déclarations de fonctions dans le *<head>* ne ralentit pas l'affichage de la page. L'exécution des fonctions est entraînée par leurs **appels**. Exemple :

```
nom_de_la_fonction(arguments);
```

### B. Arguments et paramètres

On appelle « **paramètres** » les variables présentes dans les parenthèses de la définition de la fonction. On appelle « **arguments** » les valeurs passées à la fonction lors de son appel, valeurs servant à initialiser les paramètres.

Dans l'exemple ci-dessous, la fonction *message()* est appelée et reçoit en argument la valeur « Bienvenue » qui initialise le paramètre *texte*.

```
function message(texte) {  
    alert(texte);  
}  
message("Bienvenue"); //appel de la fonction
```

En JavaScript, les arguments sont toujours passés par valeur. Cela signifie que si l'appel de la fonction passe une variable en argument, seule la valeur de cette variable sera envoyée. Cela implique que les modifications des valeurs des paramètres ne sont effectives que localement à la fonction.

Toutefois, ce n'est pas le cas si l'argument est un objet : le passage des objets en argument se fait par référence. Cela implique que les modifications au sein de la fonction ont un impact sur l'objet d'origine.

### C. Renvoyer une valeur

Parfois, on aura besoin de fonctions qui renvoient une valeur, un résultat. Pour cela, nous utiliserons le mot clef **return**. Exemple :

```
function aire(longueur, largeur) {  
    var aire = longueur * largeur;  
    return aire;  
}
```

### D. Curiosité

Voici donc une fonction appelée *aire* qui comporte deux paramètres : *longueur* et *largeur*. Commence ensuite le code de cette fonction, et là, sous nos yeux ébahis, on déclare une variable portant le **même nom que la fonction** !

Nous avons vu précédemment les règles en matière de noms de fonction et de variables, mais est-il possible de déclarer deux variables ayant le même nom ? En théorie non. Mais si ces deux variables/fonctions ne se trouvent pas dans le même sous-programme, alors c'est possible.

Explication : si on déclare une variable *aire* dans le programme principal ou dans une autre fonction, en plus de la variable *aire* déjà existante dans la fonction *aire*, c'est possible. Par contre, si on déclare une deuxième variable *aire* dans la fonction *aire*, cela provoquera des erreurs.

Dans tous les cas, il vaut mieux éviter ce genre d'écriture qui prête à confusion. De plus, veuillez toujours à choisir des noms de variables explicites : *aire* contient l'aire, *longueur* contient la longueur, etc. Si la variable contenant la longueur s'appelait *poids*, le programme fonctionnerait quand même, mais lorsque vous relirez votre code plus tard, vous risquez de ne pas vous y retrouver. Donc, pour plus de clarté, une petite correction s'impose :

```
function aire(longueur, largeur) {  
    var aire_rectangle = longueur * largeur;  
    return aire_rectangle;  
}
```

Exemple d'appel de cette fonction qui renvoie une valeur :

```
document.write(aire(4,5));
```

La fonction *aire()* est donc appelée, elle calcule la valeur de *aire\_rectangle* qui vaut  $4 * 5$  et renvoie ce résultat. Le résultat (20) est alors utilisé par la méthode *document.write()* et affiché dans la page.

## Exercice 2 : imposer la majuscule dans un champ texte

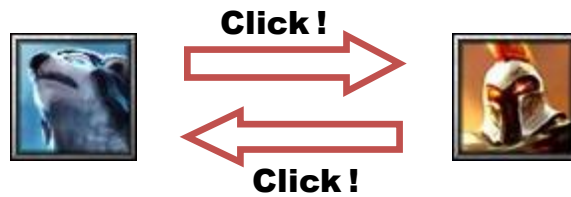
### Objectifs

Créez un champ texte HTML invitant l'utilisateur à y entrer son nom. Lorsque l'utilisateur a entré son nom et qu'il « sort » du champ, transformez la première lettre du nom en majuscule.

### Aide

1. Codez la balise `<input>` du champ texte, avec un attribut HTML écoutant l'évènement adéquat.
2. A partir de cette écoute d'évènement, appelez une fonction en passant en paramètre l'objet `<input>` déclencheur. NB : lorsque l'élément déclencheur doit être passé en paramètre, l'utilisation de l'objet *this* est indiqué.
3. Codez la déclaration de la fonction dans un élément `<script>`.
4. Dans cette fonction, récupérez la valeur du champ texte. A ce stade, il est conseillé de tenter un `alert()` pour vérifier que l'évènement choisi est le bon, que la fonction est correctement déclarée et appelée et que vous avez bien réussi à récupérer le nom de l'utilisateur. Aidez-vous de la console si besoin.
5. A l'aide des méthodes des objets *string* (voir théorie), récupérer la première lettre du mot, transformez la en majuscule, concaténez-la avec le reste du mot (sans la première lettre) et affectez le résultat à la valeur du champ.

### Exercice 3 : changer la source d'une image lors d'un clic

**Principe :**

Dans cet exercice, nous allons mettre en pratique les notions d'évènement, de fonction et de propriétés en créant une page HTML avec une image qui change de source lorsqu'on la clique.

**Démarche :**

Créez une page HTML contenant une image. Prévoyez une seconde image dans le dossier du site.

Associez à cet élément `<img>` l'écoute de l'évènement clic afin d'appeler une fonction nommée *change(this)*.

Placez un élément `<script>` dans le `<head>` de votre page et déclarez la fonction *change(image)*.

Complétez cette fonction de façon à ce qu'elle change la propriété *src* de *image* par une autre url.

**Suite de l'exercice :**

Créez une variable globale enregistrant le nombre de fois que l'image a été cliquée. Initialisez cette variable à 0.

Prévoyez une incrémentation de cette variable au bon endroit.

Dans la fonction *change()*, testez si votre variable est paire ou impaire afin d'afficher la première image, puis la seconde, puis la première à nouveau, puis la seconde, etc.

## Exercice 4 : le Morpion

### Principe :

Le jeu du morpion comporte une grille de 3x3 cases et se joue à deux (deux couleurs de pions).

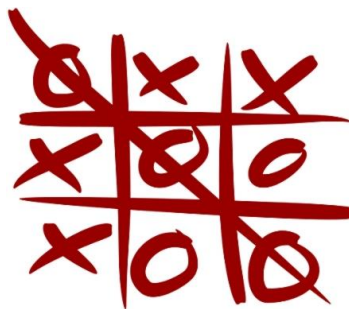
Les joueurs posent chacun à leur tour un pion dans l'une des cases. Le gagnant est celui qui arrive à aligner 3 pions de sa couleur.

Les joueurs ne peuvent pas déposer de pion dans des cases qui ne sont pas libres, ni changer leur pion de position une fois qu'ils sont déposés.

Le jeu s'arrête soit lorsqu'un joueur a gagné, soit lorsque la grille est pleine et que personne n'a gagné (égalité).

### Fonctionnalités supplémentaires :

1. Proposez un élément visuel signalant la victoire d'un des joueurs.
2. Tirez au hasard le joueur qui débute la partie.
3. Créez un élément visuel montrant « à qui est le tour ».
4. Permettez aux joueurs de faire plusieurs parties de suite, et affichez le score de chaque joueur ainsi que le nombre d'égalités.
5. Permettez au joueur de choisir un mode de jeu : soit 1 contre 1, soit 1 contre l'ordinateur.
6. Rendez votre IA la plus habile possible !



## 1.11 DOM

### A. Introduction au DOM

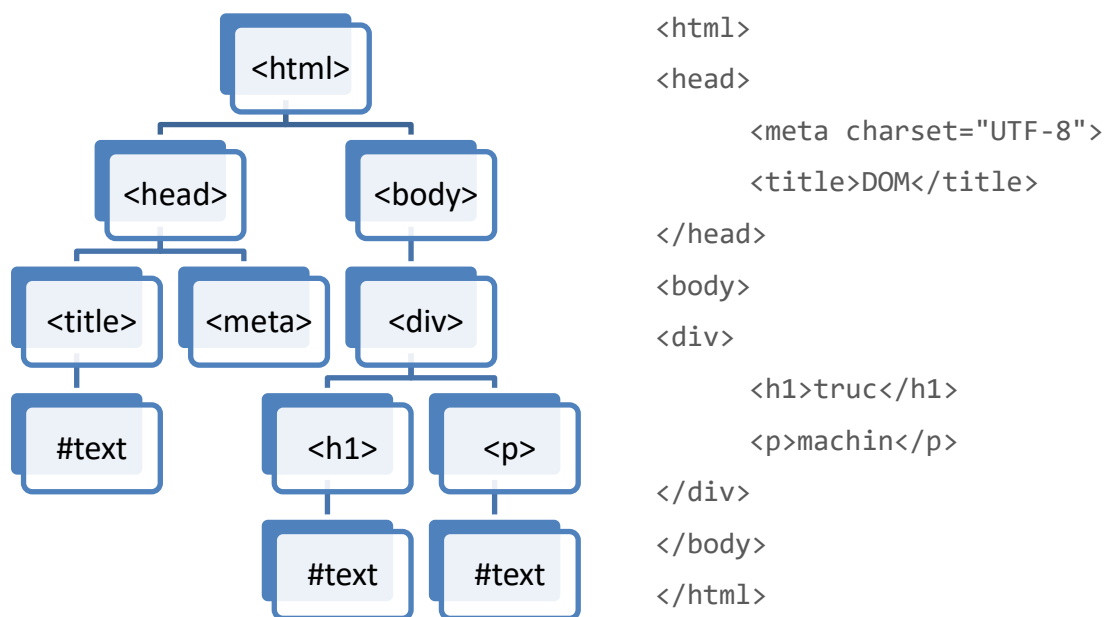
Le **Document Object Model** (DOM) est une Interface de Programmation (API) pour les langages XML et HTML.

Les API sont des ensembles d'outils permettant de faire communiquer entre eux des programmes ou des langages.

Le JavaScript est capable d'accéder au DOM pour modifier, créer ou supprimer des éléments HTML.

DOM est une technologie détenue par le W3C et qui en est actuellement à sa 3<sup>e</sup> version : DOM-3.

DOM voit les pages HTML (et XML) comme des arbres hiérarchiques où **chaque élément et chaque texte sont vus comme des nœuds**.



## B. Manipuler les éléments

<code>getElementById()</code>	Méthode permettant d'accéder à un élément via son ID.
<code>getElementsByTagName()</code>	Méthode permettant d'accéder à des éléments via leur type.
<code>getElementsByName()</code>	Méthode permettant d'accéder à des éléments par leur attribut <i>name</i> . Remarque : Attention, depuis HTML5, l'attribut <i>name</i> ne s'utilise plus que dans les éléments de formulaire.
<code>getElementsByClassName()</code>	Méthode permettant d'accéder à des éléments par leur attribut <i>class</i> . Remarque : Attention, cette méthode est incompatible avec IE8 et versions antérieures.

Exemple :

```
<div id="alpha">
  <p>Bonjour les gars !</p>
</div>
<script>
  var objet = document.getElementById('alpha');
  alert(objet);
</script>
```

Le résultat de cet exemple affichera « [objectHTMLDivElement] », signalant ainsi que la variable *objet* contient un élément HTML de type *div*.

Remarque : l'objet *objetHTMLDivElement* est un sous-objet de l'objet *HTMLDivElement*, lui-même sous-objet de l'objet *Element*, lui-même sous-objet de l'objet *Node*. Par héritage, les méthodes et propriétés des objets parents se propagent chez les objets enfants.

## B. Méthodes des noeuds

<code>childNodes()</code>	nœuds enfants
<code>firstChild()</code>	premier nœud enfant
<code>lastChild()</code>	dernier nœud enfant
<code>parentNode()</code>	nœud parent
<code>nextSibling()</code>	prochain nœud d'un type (nœud de même niveau)
<code>previousSibling()</code>	nœud précédent d'un type (nœud de même niveau)
<code>nodeName()</code>	nom du nœud
<code>nodeValue()</code>	contenu du nœud
<code>nodeType()</code>	type du nœud



## C. Modifier le contenu des éléments

La propriété ***innerHTML*** permet de récupérer et de modifier tout le contenu d'un élément HTML : aussi bien le texte que les éléments enfants.

Exemple : le code ci-dessous affichera le message « <p>Bonjour les gars !</p> » dans une boîte d'alerte.

```
<div id="alpha">
  <p>Bonjour les gars !</p>
</div>
<script>
  var contenu = document.getElementById('alpha').innerHTML;
  alert(contenu);
</script>
```

### Exercice 5 : Total = nombre \* prix à l'unité

#### Théorie

Nous avons vu comment modifier les propriétés d'un élément déclencheur (celui qui écoute l'évènement) avec le mot-clé *this*. Nous allons maintenant voir comment modifier les autres éléments de la page avec *document.getElementById()* et la propriété *.innerHTML*.

#### Commande

Vous avez commandé  article à 12.50€.

Total : 12.50€

#### Principe

Le code HTML reprend un champ de type *number* et un prix à l'unité. Lorsque le visiteur modifie le nombre dans le champ, le total s'adapte.

#### Aide

Pour manipuler des données comme le total ou le prix à l'unité, il est conseillé de les placer dans des éléments *<span>* possédant un *id*.

Dans cet exercice, il est souhaité que la valeur du prix à l'unité soit chargée à partir de la page (et ne pas directement multiplier par 12.50€).

## Exercice 6 : le Formulaire Dynamique

### Principe

Un des principaux atouts du JavaScript est sa capacité à modifier la structure HTML d'une page suivant les actions du visiteur.

Nous allons, dans cet exercice, exploiter à nouveau cet atout en créant un formulaire dans lequel des champs apparaissent et disparaissent suivant des cases cochées ou décochées et où le prix s'adapte selon un certain nombre de paramètres.

### 1<sup>re</sup> étape

Codez le formulaire HTML.

### Votre pizza

Margarita ▼  
☐ avec suppléments  
☒ sans suppléments

Paiement :

☒ CASH  
☐ Visa

Prix :

0€



### 2<sup>e</sup> étape

En JavaScript, écoutez l'évènement *onclick* sur les boutons radios « avec suppléments » et « sans suppléments ».

Déclarez une fonction recevant un paramètre *x* dont la valeur (au choix) indique si les suppléments sont visibles ou non. Dans cette fonction, on utilise *getElementById()* sur un *<div>* contenant les suppléments et un *innerHTML* pour en modifier le contenu (les anchois et les frites ne sont pas codés en HTML, mais leur code HTML est injecté depuis la fonction JavaScript).

Attention, ne choisissez pas des noms de fonctions qui soient les mêmes que les valeurs de vos attributs *name*.

### Votre pizza

Margarita ▼  
☒ avec suppléments  
☐ sans suppléments  
☐ Anchois  
☐ Frites  
☐ Olives

Paiement :

☒ CASH  
☐ Visa

Prix :

0€

### 3<sup>e</sup> étape

Le prix doit être modifié via une fonction unique, à chaque clic sur des suppléments et sur des modes de paiement. Le prix est égal au prix de la pizza (8€) + prix des suppléments (0.50€ par supplément) + 2€ si le mode de paiement est Visa.

Afin de donner des prix différents aux différents choix de pizza, créez dans cette fonction, un tableau associatif dont les indices sont les *value* des *<option>* de la liste déroulante, et dont les valeurs sont les prix des pizzas ! Par exemple « Margherita » => 7€ ; « Regina » => 8€ ; « Capricciosa » => 9€ ; « Quattro Formaggi » => 9€ ; « Diavola » => 8.50€ ; « Marinara » => 7.50€ ; ...

### Votre pizza

Capricciosa ▼  
☒ avec suppléments  
☐ sans suppléments  
☒ Anchois  
☒ Frites  
☐ Olives

Paiement :

☒ CASH  
☐ Visa

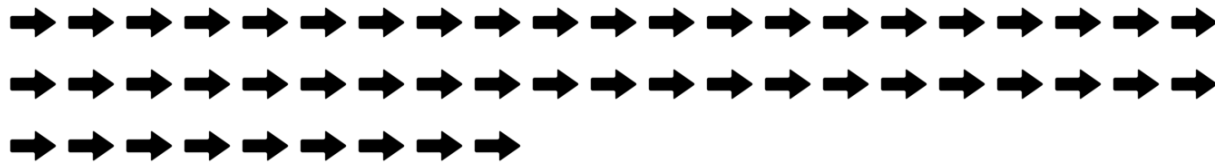
Prix :

9€

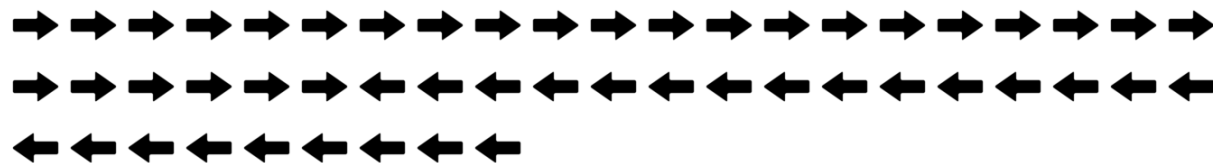
## Exercice 7 : les Flèches

### Principe

Lors du chargement de la page, une fonction JavaScript est appelée et génère entre 10 et 100 flèches pointant vers la droite.



Chaque flèche est cliquable. Un clic sur une flèche fait tourner la flèche vers la gauche et toutes les flèches suivantes vers la gauche également !



### Aide

Pour générer les flèches, vous aurez besoin d'utiliser la fonction *random()* qui, en JavaScript, a la particularité de renvoyer un nombre réel entre 0 et 1... à vous d'en faire un entier entre 10 et 100 !

Les images de flèches sont évidemment générées via une boucle. Dans des cas comme celui-ci, préférez construire une chaîne de caractère avec le « résultat » et injectez la en une fois, après la boucle. Ceci afin de ne pas appeler une centaine de fois *document.getElementById()*, ce qui ralentirait votre script.

## Chapitre 2 : jQuery

### 2.1 Introduction

jQuery est une bibliothèque (ou *framework*) JavaScript qui permet notamment de manipuler les éléments HTML afin de créer diverses animations.

Gardons à l'esprit que tout ce que nous créons avec jQuery, nous pourrions le faire en JavaScript sans utiliser cette fameuse bibliothèque. Cependant les avantages de jQuery en font un outil incontournable de la programmation web. Pourquoi ? Quels sont les **avantages** apportés par jQuery ?

#### A. Rapide

- Alors que créer un simple mouvement prendrait des dizaines de lignes de code en JavaScript, une seule ligne de code suffit en jQuery.

#### B. Compatible

- Vous l'ignorez peut-être si vous êtes un jeune webmaster, mais les problèmes de compatibilité ont longtemps été la bête noire des développeurs. jQuery est 100% compatible. Plus besoin de prévoir de longues lignes de code pour prendre en charge les différents navigateurs.

#### C. Léger

- jQuery c'est du JavaScript, c'est-à-dire... du texte. Quoi de plus léger que du texte ? La bibliothèque ne pèse même pas 100ko.

#### D. Évènementiel

- Comme jQuery est basé sur le JavaScript, il connaît tous les événements du langage : clic, survol, focus, etc.

#### E. Plugins à volonté

- Des milliers de plugins existants sont téléchargeables. Ce grand choix de plugins est une force supplémentaire car ils permettent d'étendre les fonctionnalités de base de la bibliothèque.

#### F. Gratuit

- Et en plus c'est gratuit !

## 2.2 Mise en place de jQuery

La bibliothèque jQuery est téléchargeable à l'adresse :

<http://jquery.com/>



Deux versions existent :

- la minimalisée (appelée aussi **production**) très légère mais illisible
- la standard (appelée aussi **development**) moins légère mais lisible et éditée pour les programmeurs souhaitant modifier le code de la bibliothèque

Le fichier *jquery.js* est ensuite à placer dans le dossier de votre site ou de votre application. On le place généralement dans un sous-dossier appelé *js*.

Pour pouvoir utiliser les fonctionnalités jQuery, il faut l'inclure à la page HTML comme n'importe quel fichier JavaScript :

```
<head>
<script type="text/javascript" src="js/jquery.js"></script>
</head>
```

Remarque : cette fonction *\$()* a priorité sur la fameuse fonction *window.onload()* disponible en JavaScript.

L'ordre dans lequel se déroule l'interprétation du code par un navigateur est donc le suivant :

1. Le navigateur commence à télécharger les éléments auxquels le fichier html fait référence (images, css...).
2. Le code situé dans *\$()* est exécuté dès que possible.
3. Lorsque tout est téléchargé, le code situé dans *window.onload()* est exécuté, en dernier.

## 2.3 Utilisation de jQuery

Tout appel à une fonctionnalité jQuery passe par la fonction *\$()*.

```
$(document).ready(function(){
    $('#menu').mouseenter(function() {
    $('#truc').stop().animate({height:'200px'},{duration:1000});
    });
    $('#menu').mouseleave(function() {
        $('#truc').stop().animate({height:'0px'},{duration:1000});
    });
} );
```

Nous allons maintenant analyser ce code et découvrir comment créer nos propres effets jQuery en 3 étapes :

1. **Sélectionner les éléments HTML**
2. **Ecouter les événements**
3. **Animer**

## 1. Sélectionner

Si vous connaissez le langage CSS, vous allez apprécier ce point car la sélection en jQuery a la même syntaxe qu'en CSS :

Exemple pour sélectionner les éléments dont l'*id* est menu :

```
$('#menu')
```

Exemple pour sélectionner les éléments dont la *class* est photo :

```
$('.photo')
```

Remarque : tous les sélecteurs CSS sont valables, y compris les moins utilisés comme les sélecteurs de pseudo-classes ou d'attributs.

## 2. Ecouter les évènements

Une fois un élément sélectionné, on peut y assigner un évènement :

Exemple d'écoute de l'évènement clic :

```
$('#menu').click( ) ;
```

Exemple d'écoute de l'évènement survol :

```
$('.photo').hover( , ) ;
```

Remarque : tous les évènements JavaScript sont disponibles et même plus. Voir le récapitulatif des évènements jQuery à la fin de ce chapitre.

Remarque : certains évènements prennent plusieurs paramètres. Dans le cas de *hover()*, la première action a lieu lorsqu'il est survolé, la seconde à la fin du survol.

## 3. Animer

Une fois un évènement assigné à un élément, une action (animation) peut y être associée :

Exemple de disparition (*fadeOut*) d'un élément suite à un clic :

```
$('#menu').click(function() {  
    $('#section#actu').fadeOut("slow") ;  
}) ;
```

Remarque : souvent les animations sont personnalisables, il est notamment possible de choisir le temps que dure l'animation en millisecondes ou avec des termes comme « xslow », « slow », « normal », « fast » ou « xfast ».

Exemple de variation de la hauteur des éléments lors de leur survol :

```
$('.photo').hover(  
  function() {  
    $('.photo').stop().animate({height:'400px'},{duration:1000});  
  },  
  function() {  
    $('.photo').stop().animate({height:'200px'},{duration:1000});  
  });
```

Remarque : la fonction *animate()* est différente des autres fonctions d'animation car elle permet de faire varier dans le temps n'importe quelle propriété dont la valeur est numérique. Par exemple la hauteur, la largeur, les marges, les tailles des bordures, l'opacité, etc. Elle ne peut pas par contre faire varier des propriétés dont la valeur n'est pas numérique comme la couleur ou une URL.

## 2.4 Les évènements jQuery

Associer une fonction à un évènement jQuery permet d'appeler cette fonction lors du déclenchement de l'évènement.

### A. Evènements navigateur

Evènements	Descriptions
<b>error ( fonction )</b>	Cet évènement est déclenché lorsqu'une erreur de script est détectée par le navigateur.
<b>resize ( fonction )</b>	Cet évènement se déclenche lors de la modification de la taille d'un élément.
<b>scroll ( fonction )</b>	Cet évènement se déclenche lors du défilement ( <i>scroll</i> ) d'un élément.

### B. Evènements clavier

Evènements	Descriptions
<b>keydown ( fonction )</b>	Cet évènement se déclenche lorsque l'utilisateur presse une touche du clavier et que l'élément sélectionné a le focus.
<b>keypress ( fonction )</b>	Cet évènement est semblable à l'évènement <i>keydown</i> à la différence des touches de modification, comme MAJ (SHIFT).
<b>keyup ( fonction )</b>	Cet évènement se déclenche lorsque l'utilisateur relâche une touche du clavier préalablement enfoncée et que l'élément sélectionné a le focus.

## C. Evènements souris

Evènements	Descriptions
<b>click ( fonction )</b>	Cet évènement se déclenche lors du clic de l'élément sélectionné.
<b>dblclick ( fonction )</b>	Cet évènement se déclenche lors du double clic de l'élément sélectionné.
<b>hover ( fct_over, fct_out )</b>	Associe deux fonctions à l'évènement <i>hover</i> : une fonction lors du survol et une fonction de fin de survol.
<b>mousedown ( fonction )</b>	Cet évènement se déclenche lorsque l'utilisateur presse un bouton de la souris.
<b>mouseenter ( fonction )</b>	Cet évènement se déclenche lorsque le pointeur de la souris entre dans la surface de l'élément sélectionné.
<b>mouseleave ( fonction )</b>	Cet évènement se déclenche lorsque le pointeur de la souris quitte la surface de l'élément sélectionné. Il faudra que le pointeur se déplace à nouveau dans l'élément puis en ressorte pour que l'évènement <i>mouseleave</i> soit à nouveau déclenché.
<b>mousemove ( fonction )</b>	Cet évènement se déclenche lorsque le curseur de la souris passe sur l'élément sélectionné.
<b>mouseout ( fonction )</b>	Cet évènement se déclenche lorsque le curseur de la souris quitte un élément. A l'inverse de l'évènement <i>mouseleave</i> , l'évènement <i>mouseout</i> se déclenche à chaque fois que le pointeur quitte un des éléments fils.
<b>mouseover ( fonction )</b>	Cet évènement se déclenche lorsque l'utilisateur positionne le curseur de la souris au-dessus d'un élément. A l'inverse de l'évènement <i>mouseenter</i> , l'évènement <i>mouseover</i> se déclenche à chaque fois que le pointeur survole un des éléments fils.
<b>mouseup ( fonction )</b>	Cet évènement se produit lorsque l'utilisateur relâche le bouton de la souris.



## D. Evènements chargement de documents

Evènements	Descriptions
<b>load ( fonction )</b>	Cet évènement se déclenche lorsque l'élément en question est chargé par le navigateur.
<b>ready ( fonction )</b>	<p>Cet évènement se déclenche une fois le document DOM entièrement chargé, et prêt à être parcouru et manipulé. Cette fonction est l'une des plus importantes du module d'évènement, et peut augmenter de manière importante le temps de réponse de vos applications web.</p> <p>Assurez-vous que vous n'avez pas de code pour le gestionnaire d'évènement <i>onload</i> de la balise <code>&lt;body&gt;</code>, sinon <code>\$(document).ready()</code> ne se lancera pas.</p>
<b>unload ( fonction )</b>	Cet évènement se déclenche lorsque le visiteur quitte la page en cours, pour les éléments de la sélection.

## E. Evènements formulaires

Evènements	Descriptions
<b>blur ( fonction )</b>	Cet évènement se déclenche lorsque l'élément sélectionné perd le focus.
<b>change ( fonction )</b>	Cet évènement se déclenche lorsque l'élément sélectionné perd le focus et que son contenu a été modifié.
<b>focus ( fonction )</b>	Cet évènement se déclenche lorsque l'on clique sur un champ de formulaire afin d'en modifier la valeur.
<b>select ( fonction )</b>	Cet évènement se déclenche lorsqu'une option est sélectionnée dans une liste déroulante (balise <code>&lt;select&gt;</code> ).
<b>submit ( fonction )</b>	Cet évènement est déclenché lors de la soumission d'un formulaire.

## F. Gestionnaires d'évènements

Evènements	Descriptions
<b>bind ( évènement,fct )</b>	<p>Permet de lier directement un évènement et une fonction à un élément sélectionné.</p> <p>Exemple :</p> <pre>\$('#test_bind').bind('click', function() { });</pre> <p>L'utilisation de <i>bind</i> est déconseillée lorsque le nombre d'éléments est élevé.</p> <p>L'utilisation de <i>bind</i> est également déconseillée lorsque le DOM est modifié par un script (les nouveaux éléments ne bénéficieront pas des évènements liés via <i>bind</i>).</p>
<b>delegate (sélecteur, évènement,fct )</b>	<p>Idem que <i>live()</i>, à ceci près que <i>delegate</i> permet de lier une fonction et un évènement à n'importe quel endroit dans le DOM, et plus seulement sur le document.</p>
<b>die ( type,fonction )</b>	<p>Cette fonction est l'inverse de la fonction <i>live()</i>. Elle supprime un évènement lié à un élément. Sans argument, tous les évènements seront supprimés.</p> <p>Si un type est spécifié, alors seulement les évènements de ce type seront supprimés.</p>
<b>live ( évènement,fct )</b>	<p>Lie un gestionnaire à un évènement pour les éléments sélectionnés.</p> <p>Les évènements supportés : <i>click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, keydown, keypress, keyup</i>.</p> <p>Les évènements non supportés: <i>blur, focus, mouseenter, mouseleave, change, submit</i>.</p> <p>Cette fonction est similaire à <i>bind()</i>, mais avec quelques différences :</p> <ul style="list-style-type: none"> <li>- <i>live</i> s'applique à tous les éléments répondant au moment de l'appel à la sélection, et à tous ceux qui y répondront par la suite. Par exemple si la fonction cible le click tous les <i>&lt;li&gt;</i> de la page, <i>live</i> s'appliquera aux <i>&lt;li&gt;</i> présents. Si un <i>&lt;li&gt;</i> est ajouté dans la DOM par la suite, alors l'évènement <i>click</i> sur ce nouvel élément sera aussi ciblé. Alors que pour la fonction <i>bind</i>, il faudrait la réexécuter pour inclure le nouveau <i>&lt;li&gt;</i> dans la sélection.</li> <li>- les évènements <i>live</i> ne se propagent pas de la même manière que les évènements traditionnels et ne peuvent être stoppés en utilisant <i>stopPropagation</i> ou <i>stopImmediatePropagation</i>. Par exemple soit deux évènements liés au <i>click</i>, l'un sur <i>&lt;li&gt; &lt;a&gt;</i> et l'autre sur <i>&lt;li&gt;</i>. Un clic sur le lien</li> </ul>

	<p>dans le <code>&lt;li&gt;</code> va déclencher les deux évènements. Simplement car l'appel à <code>\$(<i>"li"</i>).bind(<i>"click"</i>, fn);</code> déclenche l'évènement lors d'un click sur un <code>&lt;li&gt;</code> ou tout élément fils du <code>&lt;li&gt;</code>.</p> <p>- Les évènements <i>live</i> ne fonctionnent que lorsqu'ils ciblent des éléments avec un sélecteur. Par exemple, ce code marche: <code>\$(<i>"li a"</i>).live(...)</code> mais ce code ne marche pas : <code>\$(<i>"a"</i>, someElement).live(...)</code> ni celui-là : <code>\$(<i>"a"</i>).parent().live(...)</code>.</p>
<b>on (évènement,fct)</b>	<p>La fonction <i>on</i> remplace admirablement les fonctions <i>bind</i>, <i>delegate</i> et <i>live</i>.</p> <pre>// exemple BIND \$('#truc').bind('click', function() { }); // équivalent ON \$('#truc').on('click', function() { });  // LIVE, ancienne méthode \$('#truc').live('click', function() { }); // La même chose, version ON \$(document).on('click', '#truc', function() { });</pre>
<b>one ( type,donnees,fonction )</b>	<p>Associe une fonction à un évènement donné. La différence avec la fonction <i>bind()</i> est que la fonction associée à l'évènement ne sera exécutée au maximum une fois pour chaque élément de la sélection.</p>
<b>trigger ( type,data )</b>	<p>Déclenche un évènement particulier pour les éléments de la sélection. Cela va aussi déclencher l'action par défaut du navigateur pour ce type d'évènement (si elle existe). par exemple, utiliser le type d'évènement <i>submit</i> dans la fonction va aussi déclencher l'envoi du formulaire par le navigateur. Cette action par défaut peut être empêchée en retournant <i>false</i> dans une des fonctions associées à l'évènement pour un des éléments de la sélection.</p>
<b>unbind ( fonction,type )</b>	<p>L'opposé de la fonction <i>bind()</i>, supprime les actions associées à un évènement particulier pour les éléments de la sélection. Si aucun argument n'est spécifié, les actions liées à tous les évènements sont supprimées. Si le type est spécifié, seulement les actions liées à ce type d'évènement seront supprimées. Si une fonction est passée en second paramètre, seulement les actions du type de la fonction passée en paramètre seront supprimées.</p>

## G. Déclencher un évènement

Il est possible de déclencher l'ensemble des fonctions associées à un évènement sans que cet évènement ait été déclenché par le visiteur.

L'exemple ci-dessous affichera le message « Test : blur ! » lorsque l'élément sélectionné (dont l'*id* est *test\_blur*) perdra le focus. Mais ce message s'affichera également lorsque l'élément dont l'*id* est *test\_click* sera cliqué.

```
$(document).ready(function(){  
    $("#test_blur").blur(function() {  
        alert("Test : blur !");  
    });  
    $("#test_click").click( function(){ $("#test_blur").blur(); });  
} );
```

## 2.5 Les animations jQuery

Effets	Descriptions
<b>animate()</b>	<p>Parmi les fonctions d'animation jQuery, <i>animate</i> est la plus utilisée. Cette fonction permet de faire varier les paramètres numériques d'un élément dans le temps. Par exemple la hauteur, la taille de la police, les marges, l'interligne, etc.</p> <p>A noter que ces propriétés devront être spécifiées comme suit : <code>marginLeft</code> au lieu de <code>margin-left</code> (écriture en dos de chameau plutôt qu'écriture CSS).</p> <p>L'élément est ensuite animé depuis ses paramètres d'origine jusqu'aux nouvelles valeurs en un certain nombre de millisecondes (1500 dans l'exemple ci-dessous).</p> <p>Exemple :</p> <pre><code>\$("#effet").animate({     width: "50%",     opacity: 0.2,     marginLeft: "0.6em",     fontSize: "2em",     borderWidth: "10px" }, 1500 );</code></pre>
<b>delay()</b>	Marque un temps de pause avant de déclencher les événements suivants.
<b>fadeIn()</b>	<p>Fait apparaître les éléments sélectionnés en augmentant leur opacité.</p> <p>Exemple :</p> <pre><code>\$("p").fadeIn("fast");</code></pre>
<b>fadeOut()</b>	<p>Fait disparaître les éléments sélectionnés en diminuant leur opacité.</p> <p>Exemple :</p> <pre><code>\$("p").fadeOut("slow");</code></pre>
<b>fadeTo()</b>	<p>Permet les effets de « fade » en précisant la vitesse et l'opacité.</p> <p>Exemple :</p> <pre><code>\$("#effet").fadeTo("normal", 0.3);</code></pre>
<b>fadeToggle()</b>	<p>Alterne <i>fadeIn</i> et <i>fadeOut</i>.</p> <p>Exemple :</p> <pre><code>\$("#effet").fadeToggle("normal");</code></pre>

<b>hide()</b>	Cache les éléments sélectionnés.  Exemple : <code>\$("#effet").hide("xfast");</code>
<b>show()</b>	Affiche les éléments sélectionnés.  Exemple : <code>\$("#effet").show("xslow");</code>
<b>slideDown()</b>	Affiche les éléments sélectionnés selon un effet de glissement.  Exemple : <code>\$("#effet").slideDown("normal");</code>
<b>slideToggle()</b>	Alterne <i>slideUp</i> et <i>slideDown</i> .  Exemple : <code>\$("#effet").slideToggle("normal");</code>
<b>slideUp()</b>	Cache les éléments sélectionnés selon un effet de glissement.  Exemple : <code>\$("#effet").slideUp("normal");</code>
<b>stop()</b>	Arrête toutes les animations en cours pour les éléments sélectionnés.  Exemple : <code>\$("#effet").stop().slideUp("normal");</code>
<b>toggle()</b>	Permet de switcher entre deux fonctions à chaque clic sur les éléments de la sélection. Dès que l'un d'entre eux est cliqué, la première fonction est exécutée, et lors d'un nouveau clic, la seconde sera exécutée, puis de nouveau la première, etc.  Exemple : <code>\$('#target').toggle(function() {     alert('Nb de clicks impair !'); }, function() {     alert('Nb de clicks pair!'); });</code>

## 2.6 Ordre des animations

Par défaut, si 2 animations portent sur un même élément, elles ont lieu successivement. Si elles ont lieu sur des éléments différents, elles ont lieu simultanément.

### Animations **successives** sur un même élément

Exemple avec *slide* :

```
$("#truc").slideUp("normal").slideDown("normal");
```

Autre Exemple avec *slide* :

```
$("#truc").slideUp("normal");  
$("#truc").slideDown("normal");
```

Remarque : avec *animate()*, il est possible de faire varier plusieurs paramètres simultanément.

### Animations **simultanées** sur 2 éléments différents

Exemple de 2 animations simultanées :

```
$("#truc1").slideUp("normal");  
$("#truc2").slideUp("normal");
```

### Animations **successives** sur 2 éléments différents

Exemple de 2 animations successives :

```
$('#truc1').slideUp({duration:"nomal",queue:true,  
complete : function() {$("#truc2").slideUp("normal");}});
```

## 2.7 Modification d'attributs

Voici les fonctions permettant de modifier les valeurs des attributs.

Fonctions	Descriptions
<b>addClass ( class )</b>	<p>Ajoute une classe pour chaque élément sélectionné.</p> <p>Cet exemple montre comment changer l'aspect d'un élément lorsqu'il est survolé. En réalité, cette méthode ne change que la classe des éléments et repose sur un fichier CSS externe :</p> <pre>//code HTML &lt;ul id="liste"&gt;     &lt;li&gt;texte1&lt;/li&gt;     &lt;li&gt;texte2&lt;/li&gt;     &lt;li&gt;texte3&lt;/li&gt; &lt;/ul&gt;  //code javascript \$("#liste li").hover(     function(){\$(this).addClass('li_hover');},     function(){\$(this).removeClass('li_hover');} );</pre>
<b>attr ( nom )</b>	<p>Cette méthode permet facilement de retrouver la valeur d'une propriété du premier élément trouvé. Si l'élément ne dispose pas de l'attribut recherché, "undefined" est renvoyé.</p> <p>Exemple où la valeur de l'attribut title du premier élément de type "em" dans la page et le stocke dans la variable titre :</p> <pre>var titre=\$("#em").attr("title");</pre> <p>Exemple où l'on assigne des paires attributs/valeurs à tous les éléments de type <i>img</i> :</p> <pre>\$("#img").attr({ src: "http://jquery.com/images/hat.gif", alt: "Logo jQuery" });</pre> <p>Exemple où l'on ajoute des attributs <i>disabled</i> à certains éléments :</p> <pre>\$("#button").attr("disabled");</pre>
<b>attr ( proprietes )</b>	Permet d'assigner un ensemble de paires clé/valeur aux éléments trouvés.



<b>attr ( valeur, clé )</b>	Assigne une paire attribut/valeur à tous les éléments concernés.
<b>hasClass ( classe )</b>	Retourne vrai si la classe spécifiée est présente pour au moins un des éléments ciblés. Equivaut à la fonction is(".maClasse").
<b>html ( )</b>	Récupère le contenu du premier élément trouvé. Ne fonctionne pas sur les documents XML (à l'exception des documents XHTML).
<b>html ( valeur )</b>	Permet de modifier le contenu d'un élément.
<b>removeAttr ( nom )</b>	Supprime un attribut des éléments concernés.
<b>removeClass ( classe )</b>	Permet de supprimer les classes spécifiées des éléments concernés.
<b>text ( )</b>	Retourne le contenu texte de tous les éléments concernés par la recherche. Le résultat est une chaîne de caractères contenant la concaténation de tous les contenus texte des éléments. Cette méthode marche avec les documents HTML et XML.
<b>text ( valeur )</b>	Assigne un contenu texte aux éléments concernés par la recherche.
<b>toggleClass ( classe )</b>	Ajoute une classe aux éléments spécifiés, la supprime si elle est déjà présente.
<b>toggleClass ( class, switch )</b>	Ajoute la classe spécifiée si switch vaut TRUE, la supprime si switch vaut FALSE.
<b>val ( )</b>	Récupère le contenu de l'attribut "value" du premier élément de la sélection. Faites attention quand vous utilisez cette fonction pour récupérer la valeur d'éléments de formulaire de type select multiple parce qu'ils peuvent recevoir un tableau de valeurs.
<b>val ( valeur )</b>	Assigne une nouvelle valeur à l'attribut "value" des éléments de la sélection. Coche, ou sélectionne, tous les éléments radio, les checkbox, et sélectionne les options correspondant à l'ensemble des valeurs envoyées.

## 2.8 Exemple d'utilisation d'AJAX avec jQuery

**Ajax** (*Asynchronous JavaScript and XML*), est une méthode de programmation basée principalement sur le JavaScript et le XML. Ajax permet d'envoyer des requêtes HTTP asynchrones au serveur, c'est-à-dire des requêtes ne demandant pas le chargement d'une page ou d'un fichier, mais demandant seulement l'envoi de quelques informations.

Ajax permet donc, lorsque la quantité de données est conséquente, de ne pas toutes les charger mais de seulement charger les données utiles selon les actions de l'utilisateur. Par exemple sur le site Google Maps, les informations utiles sont chargées lorsque l'utilisateur se déplace sur la carte.

Ajax permet aussi de mettre à jour automatiquement les données sans que le visiteur ne doivent charger la page. Par exemple les valves électroniques de l'école (c'est une page Web) se mettent à jour à chaque nouvelle info encodée, sans qu'un utilisateur doive recharger la page.

jQuery propose un ensemble de fonctions qui permettent une utilisation facile d'AJAX. Voici un exemple où l'on souhaite afficher le contenu d'un fichier externe (test.html) dans un conteneur :

```
//code javascript
$.ajax({
  type: "GET",
  url: "test.html",
  error:function(msg){
    alert( "Erreur !: " + msg );
  },
  success:function(data){
    //affiche le contenu du fichier dans le conteneur
    $('#contenu_fichier_ajax').text(data);
  });
```

## Exercice 8 : les Véhicules

### Principe

Dans cet exercice, un nombre aléatoire de véhicules (entre 80 et 160) sont générés et affichés à l'écran lors du chargement de la page. Chaque véhicule est lui-même pioché aléatoirement parmi une vingtaine d'images (selon le nombre d'images dont vous disposez) de voitures, d'hélicoptères, de robots et d'avions.

Chaque image est cliquable : au premier clic, l'image devient bleue, au clic suivant mauve, au clic suivant rouge, et puis à nouveau bleue et ainsi de suite.

A chaque clic, le « véhiculomètre » en haut s'adapte en 1000ms et représente les pourcentages de véhicules de chaque sorte (en fonction des clics de l'utilisateur). Puisque ce mouvement n'est pas instantané, il est conseillé d'utiliser jQuery.

Remarquez qu'à aucun moment, on ne vérifie si l'utilisateur a raison dans ses choix (voir l'exemple).



### Aide

Il est conseillé de travailler avec un tableau contenant une valeur pour chaque image et de l'incrémenter à chaque clic.

Pour rappel, une ligne de jQuery s'intègre très bien au sein d'une fonction JavaScript : n'oubliez pas qu'il s'agit du même langage !

## Exercice 9 : la Bubulle

Nous allons créer une navigation horizontale avec une bulle qui se déplace suivant le menu survolé et qui disparaît lorsque l'on ne survole plus le menu.



**1<sup>re</sup> étape :** réaliser une barre de navigation horizontale en HTML + CSS. Respectez les points suivants :

- 1) La barre de navigation doit faire 800px de large.
- 2) La navigation comporte 5 menus.
- 3) La structure HTML est uniquement composée des éléments `<nav>`, `<ul>`, `<li>` et `<a>`.
- 4) Le `<nav>` a une couleur de fond mais pas les autres éléments.

**2<sup>e</sup> étape :** créer la bulle.

- 1) La bulle est une balise `<div id="bulle">` qui se trouve dans `<nav>`, frère de l'élément `<ul>`.
- 2) La bulle va devoir se déplacer horizontalement. Pour cela, nous ferons varier sa propriété `left`. Choisissez donc un positionnement CSS qui utilise cette propriété (car sans positionnement, `left` n'a pas de sens).
- 3) Choisissez une couleur de fond pour la bulle, des dimensions un peu plus larges et hautes que la barre de navigation et une transparence de 50%.
- 4) Vérifiez que la bulle se trouve bien derrière les menus. Si pas, forcez-la en utilisant des positions et `z-index` !

**3<sup>e</sup> étape :** jQuery

- 1) En jQuery, animez la bulle en détectant l'entrée et la sortie dans la barre de navigation et associez à ces événements les animations `fadeIn()` et `fadeOut()`.
- 2) En jQuery toujours, détectez le survol des différents menus et animez la bulle avec `animate()` pour faire en sorte qu'elle se déplace à chaque fois que le curseur change de menu.
- 3) Personnalisez ces animations en choisissant le temps et le type de mouvement (`easing`) pour `animate()`.

## Exercice 10 : Consonnes et Voyelles

### Consonnes et Voyelles



#### Objectifs

Au chargement de page, les mots « Consonnes » et « Voyelles » du titre sont cliquables. Chaque clic génère une lettre (sans animations) aléatoire. Procédez en déclarant 1 tableau de consonnes et un tableau de voyelles parmi lesquels vous piochez des lettres.

### Consonnes et Voyelles



La seconde étape survient une fois que les 8 lettres sont choisies (pas avant). Les 2 mots du haut ne produisent plus de lettre lorsqu'ils sont cliqués. Par contre, les 8 lettres sont alors cliquables et chaque clic, fait apparaître la lettre cliquée avec une animation jQuery : le bloc se déplace verticalement vers le haut.

Une fois une lettre cliquée, elle change de couleur et d'autres clics sur cette lettre n'auront aucun effet. Pour cela, procédez avec un tableau de 8 valeurs indiquant si une lettre a déjà été cliquée ou non.

## Exercice 5 : la bulle

Nous allons créer une navigation horizontale avec une bulle qui se déplace suivant le menu survolé et qui disparaît lorsque l'on ne survole plus le menu.

**1<sup>re</sup> étape** : réaliser une barre de navigation horizontale en HTML + CSS. Respectez les points suivants :

- 5) La barre de navigation doit faire 800px de large.
- 6) La navigation comporte 5 menus.
- 7) La structure HTML doit être composée uniquement des balises : `<nav>`, `<ul>`, `<li>` et `<a>`.
- 8) Le `<nav>`, qui est le parent des autres éléments a une couleur de fond mais pas les autres éléments.

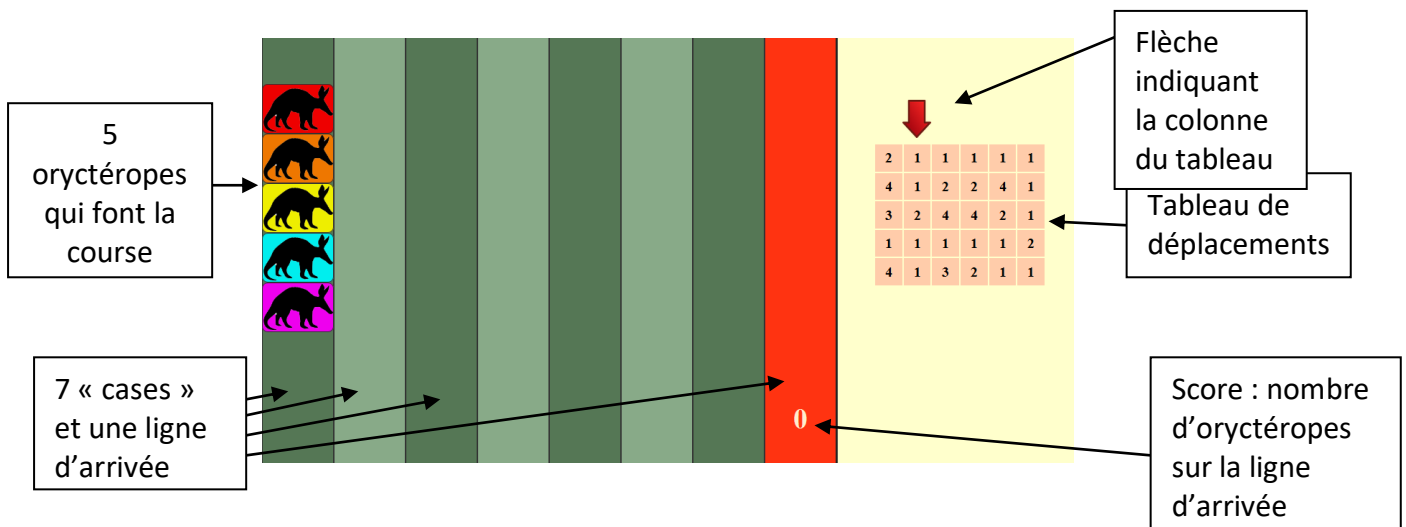
**2<sup>e</sup> étape** : créer la bulle.

- 5) La bulle est une balise `<div id="bulle">` qui se trouve dans `<nav>`.
- 6) La bulle va devoir se déplacer de gauche à droite. Pour cela nous ferons varier sa propriété *left*. Choisissez donc un positionnement CSS qui utilise cette propriété (car sans positionnement, *left* n'a pas de sens).
- 7) Choisissez une couleur de fond pour la bulle, des dimensions un peu plus larges et hautes que la barre de navigation et une transparence de 50%.
- 8) Vérifiez que la bulle se trouve bien derrière les menus. Si pas, forcez-la !

**3<sup>e</sup> étape** : jQuery

- 4) En jQuery, animez la bulle en détectant l'entrée et la sortie dans la barre de navigation et associez à ces événements les animations *fadeIn()* et *fadeOut()*.
- 5) En jQuery toujours, détectez le survol des différents menus et animez la bulle avec *animate()* pour faire en sorte qu'elle se déplace à chaque fois que le curseur change de menu.
- 6) Personnalisez ces animations en choisissant le temps et le type de mouvement (*easing*) pour *animate()*.

## Examen de labo juin 2013 : la course d'oryctéropes



### Objectifs :

Au chargement de page, une fonction JavaScript génère un tableau de 6 lignes et 5 colonnes contenant des valeurs aléatoires comprises entre 1 et 4. De plus une flèche verticale pointe sur une des colonnes du tableau.

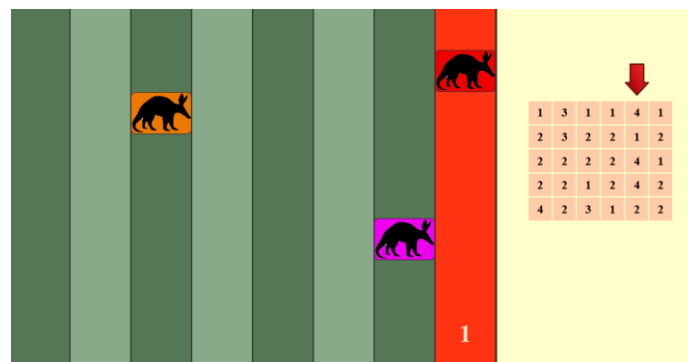
Le joueur peut cliquer sur un de ses 5 oryctéropes, celui-ci se déplace (mouvement animé avec jQuery) du nombre de case figurant dans la colonne du tableau qui est sous la flèche et dans la ligne correspondant à l'oryctérope cliqué. Par exemple, si l'on regarde le *screenshot* ci-dessous, si le joueur clique sur l'oryctérope orange (le 2<sup>e</sup>), il avancera d'une case ; s'il clique sur l'oryctérope magenta (le dernier), il avancera de 2 cases.

Juste après le déplacement de l'oryctérope, la flèche se déplace aléatoirement (mouvement animé avec jQuery) au-dessus d'une colonne du tableau.

Le score représente le nombre d'oryctéropes placés sur la ligne d'arrivée.

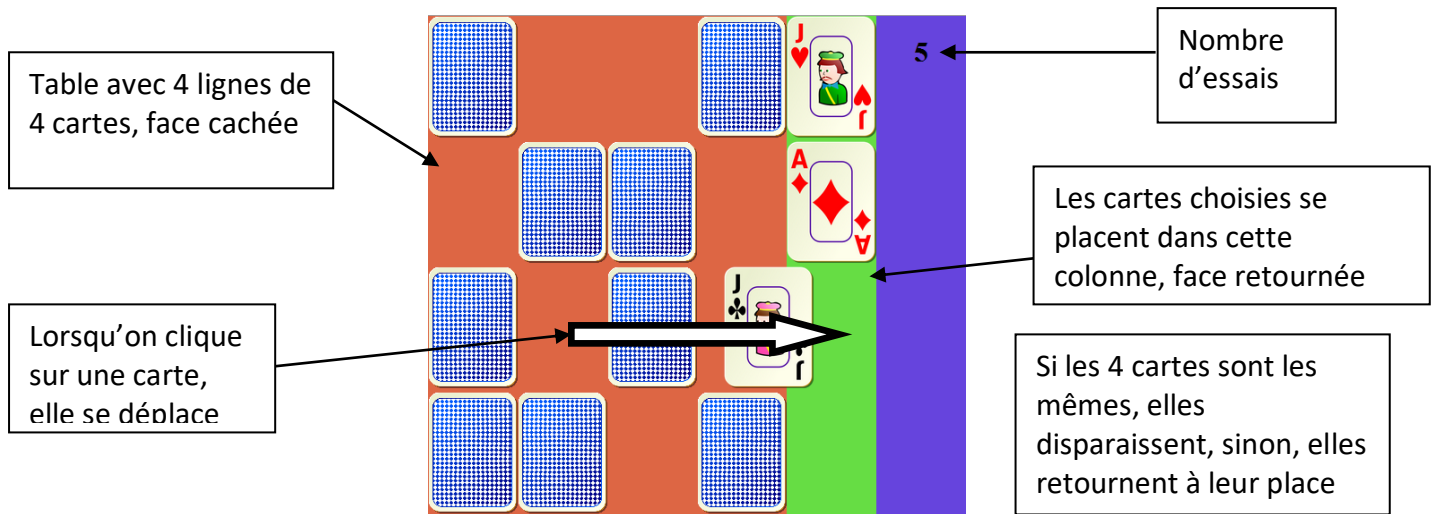
Si un oryctérope dépasse la ligne, il disparaît dans les abîmes (mouvement animé avec jQuery).

Les oryctéropes sur la ligne et ceux qui ont disparu dans les abîmes ne sont pas cliquables.



Lorsque aucun oryctérope n'est cliquable (c'est-à-dire qu'ils sont tous sur la ligne d'arrivée ou tombé dans les abîmes), le jeu se termine et un message d'alerte apparait et affiche le score.

## Examen de labo juin 2013 : Square-memory



### Objectifs :

Au chargement de page, une fonction JavaScript génère la position des 16 cartes (bien que non visibles puisque les cartes sont retournées). Attention, la 1<sup>ère</sup> ligne ne contient que des cœurs, la 2<sup>ème</sup> que des carreaux, la 3<sup>ème</sup> que des trèfles et la 4<sup>ème</sup> que des piques. Toutes les cartes sont différentes.

Le joueur doit cliquer sur 4 cartes. Lorsqu'il clique une carte d'une ligne, il ne peut pas en cliquer une autre de la même ligne. Lorsqu'une carte est cliquée, elle se tourne face visible et se déplace (effet jQuery) vers la colonne de droite.

Lorsque les 4 cartes sont choisies, le nombre d'essais s'incrémente (visible dans la page) et une vérification a lieu pour déterminer si les 4 valeurs sont les mêmes (4 valets ou 4 dames par exemple). Si le joueur a obtenu un carré, les 4 cartes disparaissent (effet jQuery), sinon elles retournent à leur position (effet jQuery) et sont tournées face cachée.









## Chapitre 3 : CSS3

### 3.1 Compatibilité de CSS3

Les normes **CSS3** sont encore actuellement en cours de réalisation et leur support par les navigateurs, bien que partiel, évolue sans cesse. Avant d'utiliser une propriété CSS3, il convient donc de toujours vérifier sa compatibilité.

A ce sujet, le site [www.w3schools.com](http://www.w3schools.com) s'avère très utile. Pour une propriété comme ***border-radius***, voici ce qu'on peut apprendre :

Property						
border-radius	5.0 4.0 -webkit-	12.0	9.0	4.0 3.0 -moz-	5.0 3.1 -webkit-	10.5

Il arrive que des navigateurs ne supportent pas à 100% une nouvelle propriété **CSS3**, et développent un support partiel de cette propriété accessible via un préfixe. On constate ci-dessus que Chrome dispose d'un support partiel de la propriété entre ses versions 4 et 5. Alors qu'Internet Explorer est directement passé au support de la norme officielle lors de sa version 9.0.

Dans ce cas, pour assurer un support le plus complet possible de cette règle, on codera :

```
... {
    -webkit-border-radius:8px ;
    -moz-border-radius:8px ;
    border-radius:8px ;
}
```

De cette manière, les visiteurs utilisant Firefox 3.6 pourront profiter de nos belles bordures arrondies. Dans la pratique, on prend rarement autant de précaution pour une règle comme ***border-radius***, mais si la règle est plus importante ou plus récente, on peut se donner la peine de coder les règles avec préfixes.

**Remarquez que l'on code toujours les règles avec préfixe avant les règles sans préfixes !**

Remarquez qu'il ne sert à rien d'utiliser ***-ie-border-radius*** ou ***-o-border-radius*** : on voit sur le site de W3Schools que cela n'existe pas.

Vu les problèmes de compatibilité qu'elles entraînent et vu la vitesse à laquelle leur support évolue, ce syllabus ne reprend pas toutes les propriétés CSS3. N'hésitez cependant pas à fouiller parmi les autres propriétés CSS3 qui ne sont pas présentées dans ce syllabus pour agrémenter vos sites.

Gardez à l'esprit qu'**une propriété non compatible peut tout de même être utilisée pour des actions non critiques de votre site comme des mouvements ou des interactions tant que l'accessibilité et l'utilisabilité de votre site n'en est pas affectée.**

Si l'on reprend l'exemple ci-dessus (qui est supporté par les navigateurs actuels), on constate que si un visiteur possède un ancien navigateur, il pourra tout de même visiter le site et ne se rendra sans doute pas compte qu'il manque des coins arrondis à quelques blocs... Il n'est donc pas erroné de coder simplement :

```
... {  
    border-radius:8px ;  
}
```

## 3.2 CSS3 comme alternative à jQuery

Les langages web, en évoluant, ont tendance à empiéter les uns sur les autres. Ainsi, le CSS3 avec ses propriétés d'animation et de transition, s'approprie une partie des utilisations de jQuery en permettant aux développeurs d'obtenir de petites animations sans utiliser de bibliothèque JavaScript.

### A. Transitions

**transition-property** : Définit les propriétés CSS qui subiront une transition. Par exemple, les marges, la largeur ou la couleur.

**transition-duration** : Définit la durée de la transition.

**transition-timing-function** : Définit la vitesse (facultatif) et peut prendre les valeurs *linear*, *ease*, *ease-in*, *ease-out*, *ease-in-out*.

**transition-delay** : Définit l'avance ou le retard que prendra la transition vis à vis de son déclenchement (facultatif).

Comme pour les autres propriétés multiples, il est possible de spécifier les 4 propriétés en une seule ligne.

Exemple :

```
a {  
    color: #f00;  
    transition-property: color;  
    transition-duration: 1s;  
}  
a:hover, a :focus {  
    color: #f0f;  
}
```

### B. Animations et *keyframes*

```
@keyframes move1 {  
    from {left: 0px;}  
    to {left: 240px;}  
}  
  
#blocQuiBouge {  
    animation: move1 8s infinite;  
}
```

### C. Comment choisir entre CSS3 et jQuery ?

CRITERES	MEILLEUR CHOIX
FACILE A DEVELOPPER	CSS3
COMPATIBLE	jQuery
LEGER	CSS3
EVENEMENTIEL	jQuery
ANIMATION DE LA COULEUR	CSS3
ANIMATION D'UN ELEMENT NON DESCENDANT DE L'ELEMENT DECLENCHEUR	jQuery

## Chapitre 4 : responsive design

### 4.1 Introduction

La complexité de l'agencement des informations dans un site, due à la grande variété de résolutions d'écran, a empiré avec l'arrivée des Smartphones et des tablettes. La solution qui consistait à développer un site dont le contenu se limitait à un bloc centré d'une largeur définie (par exemple 960px) passant sur la plupart des résolutions, ne convient plus à la situation actuelle.

Actuellement, la conception d'un site web responsive nécessite d'imaginer l'agencement des informations sur des tailles d'écran allant de 320px à plus de 1600px. Mais est-on obligé de développer Responsive ?

### 4.2 Site dédié au mobile, application mobile ou site responsive ?

Les trois solutions actuelles répondant à cette problématique sont le **site dédié**, l'**application mobile** et le **site responsive**. Chacune de ces solutions a des avantages et des inconvénients dont voici un petit résumé.

#### Le site dédié

La solution du site dédié consiste à développer plusieurs sites hébergés à différentes adresses. Par exemple un site principal pour Desktop, une version mobile du même site (avec sans doute un peu moins de contenus et une URL légèrement différente comme [www.monsite.be/mobile](http://www.monsite.be/mobile)) et peut-être même un 3<sup>e</sup> site réservé aux tablettes.

Un script PHP est habituellement utilisé pour détecter l'appareil du client et le diriger vers la bonne URL.

Parmi les **avantages** de cette solution, citons la possibilité d'affiner les contenus, la structure et la mise en page du site selon le périphérique visé. Ceci s'avère d'autant plus nécessaire lorsqu'il s'agit d'événements : le survol, le *drag&drop*, le *touch*, sont dorénavant des événements réagissant différemment selon les périphériques.

Précisons également que le site dédié est souvent la solution la plus rapide et la moins coûteuse pour un site existant souhaitant s'adapter à cette nouvelle mode du mobile.

Parmi les **désavantages**, n'oublions pas qu'un site dupliqué demandera plus de temps de maintenance qu'un site unique. Autre inconvénient : les détections d'appareil pour rediriger le visiteur sur la bonne URL ne sont pas à 100% fiables

#### L'application native

L'application native est développée dans des langages spécifiques aux appareils ciblés : iOS, Android, WindowsPhone, etc. et se télécharge depuis un "Store" (AppStore, Google Play, Windows store).

L'application mobile est installée sur le périphérique du client, garantissant ainsi une optimisation de l'adaptation de l'interface au périphérique et permettant de prendre en charge les fonctionnalités natives (*touch*, accéléromètre, GPS, appareil photo, etc.) avec plus de facilité.

Malheureusement, cette solution demande un développement spécifique dans plusieurs langages (propres à iOS, Android, WindowsPhone, etc.) donc un coût de développement et de maintenance plus élevé, sans parler du coût des licences...

Parmi les autres inconvénients, précisons que l'application mobile a un contenu non indexable par les moteurs de recherche classiques et que leur mise à jour nécessite une action de l'utilisateur.

## Une version responsive

Plus compliqué et plus long à mettre en place qu'un site unique non adapté (environ 25% de travail supplémentaire) mais tout de même plus rapide que plusieurs sites (solution du site mobile), le site Responsive s'annonce pourtant comme la solution idéale : une seule URL (meilleur référencement, pas de problème de *duplicate-content*), une maintenance plus rapide, des coûts et des délais généralement inférieurs aux 2 autres, une mise à jour immédiate, un déploiement multiplateformes, et la possibilité de bidouiller un site actuel pour le rendre Responsive (pas toujours facile, mais faisable).

Cependant, développer un site Responsive demande de bonnes connaissances en CSS, une meilleure planification du projet (plusieurs maquettes de différentes tailles) et veille technologique constante.

Pour finir, n'oubliez pas que depuis avril 2014, Google pénalise (en matière de référencement), les sites inadaptés aux mobiles, une solution de site dédié ou de site responsive devient dès lors incontournable.

## 4.3 Statique, fluide, adaptatif ou *responsive* ?

Au sens large du terme, un site *responsive* est donc un site qui s'adapte à toutes les résolutions d'écran à partir de 320px. Mais si nous souhaitons être rigoureux, il convient de distinguer les sites web **statiques**, **liquides**, **adaptatifs** et ***responsive***.

### Le design statique

Un design **statique** (ou fixe) se réfère à des dimensions figées (par exemple 960px) quelle que soit la surface de l'écran. La grande majorité des sites web était construite sur cette base avant l'arrivée du Responsive Web Design dans les années 2010.

### Le design fluide

Un site web **Fluide** (ou liquide) est un site web dont toutes les largeurs sont exprimées en unités variables (pourcentages, em, vw, etc.) et qui s'adapte généralement automatiquement à la taille de fenêtre, jusqu'à une certaine mesure.

### Le design adaptatif

Un design **Adaptatif** est une amélioration du design statique : les unités de largeur sont fixes, mais différentes selon la taille de l'écran, qui est détectée via les *media-queries*. Cette solution tient compte des principaux points de rupture (320px, 480px, 768px, 1024px, etc.) et adapte le gabarit en conséquence.

### Le design *responsive*

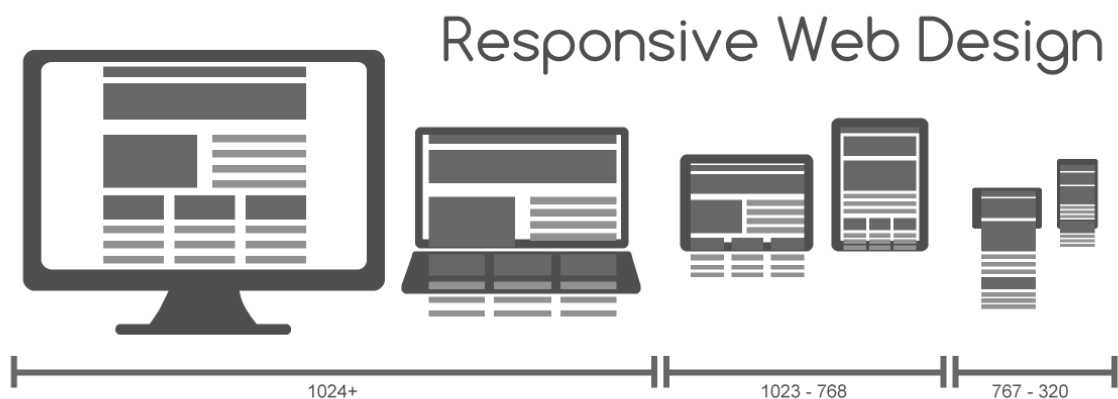
Un site web **Responsive** est une combinaison judicieuse de design fluide et adaptatif. Les conteneurs principaux s'adaptent proportionnellement à toutes les dimensions (=fluide) et certains points de rupture provoquent des changements significatifs d'agencement de contenus (=adaptatif) : souvent des blocs côte à côte qui se retrouvent l'un en dessous de l'autre, ou une répartition en 3 ou 4 colonnes qui devient une répartition en 2 colonnes.

## 4.4 En pratique

Commencez par coder la méta-balise *viewport* qui permet de faire la différence entre la résolution réelle (souvent élevée sur les Smartphone) et la résolution apparente (plus petite) :

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Lors de la conception de votre site, prévoyez toujours au minimum 2 maquettes afin de visualiser comment vos contenus s'agenceront sur des résolutions plus petites.



### Conseils CSS

Lorsque vous codez le CSS de votre site, évitez de préciser des dimensions si ce n'est pas nécessaire.

Privilégiez **width:auto** ; plutôt que **width:100%** ; qui a tendance à dépasser si l'élément comporte un *padding* ou un *border*.

Pour les images, un bon truc est de coder : **img { max-width:100% ; }** afin qu'aucune image ne sorte de son conteneur.

Enfin, travaillez avec le débogueur pour chasser les éléments qui provoquent l'apparition de la barre de scroll horizontal. Un design 100% responsive ne comporte qu'une seule barre de scroll !

Voici un exemple de media-query qui vous permettra de détecter les « points de rupture ». Prévoyez-en autant que vous souhaitez et codez les à la fin de votre fichier CSS. Il n'est pas question de recoder tout le CSS dans chaque media-query mais de ne coder que les instructions contredisant le CSS principal.

```
@media screen and (max-width: 760px) {  
    #main { width :760px ; }  
    ...  
}
```

## 4.5 FlexBox

### Introduction

FlexBox est une nouveauté CSS3 s'annonçant comme une solution de remplacement idéale pour les éléments de type bloc. FlexBox apporte de nouvelles possibilités de mise en page mais résout surtout certaines difficultés de mise en page dues aux nouvelles résolutions d'écran.

Les FlexBox, appelées aussi **boîtes flexibles**, apportent ces 4 nouveaux avantages :

1. **Distribution** : les éléments sont présentés horizontalement ou verticalement (et passent de l'un à l'autre si la résolution d'écran l'exige), avec passage à la ligne autorisé ou non
2. **Alignement** : horizontal vertical, justifié ou réparti
3. **Réorganisation des éléments** : indépendamment du flux
4. **Fluidité** : gestion des espaces disponibles

### Fonctionnement

FlexBox se fonde sur une architecture simple :

- Un **flex-container** désigné par la déclaration CSS : **display : flex ;** (ou *inline-flex ;*)
- Des **flex-item** qui sont **automatiquement** les enfants du *flex-container*

Un élément *flex-item* n'est ni un bloc ni une ligne. D'ailleurs les valeurs de **display** autres que **none**, et certaines propriétés de positionnement comme **float** sont sans effet sur lui.

Pour en savoir plus sur la compatibilité de FlexBox, n'hésitez pas à consulter CaniUse.com.

### Axe de distribution (*flex-direction*)

L'axe de distribution des éléments **flex-items** se définit par la propriété **flex-direction** appliquée au **flex-container**.

Les valeurs possibles de **flex-direction** sont :

- **row** : distribution horizontale (valeur par défaut)
- **row-reverse** : distribution horizontale en sens inverse du flux
- **column** : distribution verticale
- **column-reverse** : distribution verticale en sens inverse du flux

### Permission de retour à la ligne (*flex-wrap*)

La propriété **flex-wrap** définit si les flex-items peuvent passer à la ligne ou pas. Ce passage à la ligne devient un passage à la colonne si l'axe de distribution est vertical.

Les valeurs possibles de **flex-wrap** sont :

- **nowrap** : les éléments ne passent pas à la ligne (valeur par défaut)
- **wrap** : les éléments passent à la ligne
- **wrap-reverse** : les éléments passent à la ligne dans le sens inverse



## Alignement selon l'axe principal (*justify-content*)

Toujours appliqué au ***flex-container***, la propriété ***justify-content*** a pour valeurs possibles :

- ***flex-start*** : éléments positionnés au début du sens de lecture (valeur par défaut)
- ***flex-end*** : éléments positionnés à la fin
- ***center*** : position centrale
- ***space-between*** : répartition justifiée avec espaces entre
- ***space-around*** : variante de répartition justifiée avec espaces au début, à la fin et entre

## Alignement selon l'axe secondaire (*align-items*)

Suivant l'axe secondaire (vertical si l'axe principal est horizontal, et inversement), les alignements sont définis par la propriété ***align-items***, dont les valeurs possibles sont :

- ***stretch*** : étirés dans l'espace disponible (valeur par défaut)
- ***flex-start*** : au début
- ***flex-end*** : à la fin
- ***center*** : au centre
- ***baseline*** : généralement identique à ***flex-start***

## Alignement particulier (*align-self*)

La propriété ***align-self***, permet de distinguer l'alignement d'un flex-item de ses frères. Les valeurs de cette propriété sont identiques à celles de ***align-items*** mais cette propriété s'applique cette fois au(x) ***flex-items*** concernés.

## Propriété margin

Appliquée à des ***flex-items***, la propriété ***margin*** permet de centrer verticalement et horizontalement avec un simple ***margin:auto*** ; ce qui n'est pas possible avec d'autres types d'éléments.

Il est également possible d'aligner un ***flex-item*** dans le bas de son ***flex-container*** avec ***margin-top:auto*** ;

## Ordonnancement

Un autre avantage des FlexBox est de pouvoir réordonner les éléments grâce à la propriété ***order***. Cette propriété a un fonctionnement semblable à celui de ***z-index*** : des valeurs numériques sont attribuées aux éléments et ils sont triés selon ces valeurs en ordre croissant.

## Flexibilité

La propriété **flex** s'applique aux **flex-items** et comprend 3 sous-propriétés : **flex-grow**, **flex-shrink** et **flex-basis**, et dont les fonctionnalités sont:

- **flex-grow** : capacité d'un élément à s'étirer dans l'espace restant (par défaut : 0)
- **flex-shrink** : capacité d'un élément à se contracter si nécessaire (par défaut : 1)
- **flex-basis** : taille initiale de l'élément avant que l'espace restant ne soit distribué (par défaut : auto)

Par défaut, les **flex-items** n'occupent que la taille minimale de leur contenu, mais on peut rendre cette taille flexible, c'est-à-dire lui permettre de s'étendre afin d'occuper l'espace libre de son **flex-container**. Ceci se fait avec la déclaration **flex : 1** ; (ou **flex-grow : 1** ;) appliquée aux **flex-items**.

Il est possible de donner des valeurs différentes aux **flex-items** pour, par exemple, qu'un élément prenne 3 fois plus de place qu'un autre : **flex :3** ;

## Les flottants et les FlexBox

Les éléments flottants ont un comportement un peu différent lorsqu'ils côtoient des FlexBox :

- Les flottants ne débordent pas des **flex-container** ou des **flex-items**
- Les **flex-container** et les **flex-items** ne coulent pas autour des flottants (affichage de type article de journal...)
- Les marges ne fusionnent pas

## 4.6 Plus d'information sur ce sujet ?

Voici certaines sources ayant servi à la rédaction de ce chapitre :

<http://www.alsacreations.com/article/lire/1615-cest-quoi-le-responsive-web-design.html>

<http://www.alsacreations.com/article/lire/1559-responsive-web-design-present-futur-adaptation-mobile.html>

<http://www.alsacreations.com/tuto/lire/1493-css3-flexbox-layout-module.html>

# Récapitulatifs

## Récapitulatif HTML

### Code de base de tout document HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>...</title>
</head>
<body>
</body>
</html>
```

### Méta-balises

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<meta name="description" content="As des as du FBI, Bill Baroud est
un espion solide, un homme moulé dans de l'acier, ...">
<meta name="keywords" content="FBI, espion, Bill Baroud, héros">
<meta name="author" lang="fr" content="Bill Baroud">
<meta name="robots" content="index, follow">
```

### Titres et éléments de texte

```
<h1>Titre de la page (unique)</h1>
<h2>Titre de deuxième niveau</h2>
<h3>Titre de troisième niveau</h3>
<h4>Titre de quatrième niveau</h4>
<p>Paragraphe</p>
<strong>élément de texte important</strong>
<blockquote>Citation (bloc)</blockquote>
<cite>Citation (ligne)</cite>
<address>Citation</address>
<acronym title="Federal Bureau of Investigation">FBI</acronym>
```

### Tableaux

```
<table>
  <tr>
    <td rowspan="2">Cellule avec fusion vertic.</td>
    <th colspan="2">Cellule d'en-tête avec fusion horiz.</th>
  </tr>
  <tr>
    <td>Cellule</td>
    <td>Cellule</td>
  </tr>
```

```
</table>
```

### Listes (<ul> : non numérotée ; <ol> : numérotée)

```
<ul>
  <li>Elem1
  <li>Elem2
</ul>
```

### Formulaires

```
<form action="form.html" method="post">
  <fieldset>
    <legend>identité</legend>
    <label>Un champ texte classique : <input type="text"
name="prenom" size="32"></label>
    <label>Un champ texte avec auto-complétion : <input
type="text" name="nom" size="32" list="noms"></label>
    <datalist id="noms">
      <option value="Danes">
      <option value="Lewis">
      <option value="Patinkin">
    </datalist>
    <textarea cols="50" rows="5" name="commentaire">Un champ
texte multilignes</textarea>
    <label>Password : <input type="password" name="password"
size="32"></label>
    <label>Un nombre : <input type="number" name="enfants"
value="18"></label>
    <label>Un email : <input type="email" name="email"
size="32"></label>
    <label>Une date: <input type="date"
name="naissance"></label>
    <label><input type="checkbox" name="case1">Une case à
cocher</label>
    <label><input type="radio" name="choixunique" checked>Un
bouton radio déjà coché</label>
    <label><input type="radio" name="choixunique">Un bouton
radio</label>
    <select>
      <option value="B" selected>Belgique
      <option value="F">France
    </select>
    <input type="submit" value="Valider" name="submitted">
  </fieldset>
</form>
```

### Retours à la ligne et séparations horizontales

```
<br>
<hr>
```

**Liens**

```
<a href="dossier/page.html">Lien interne</a>
<a href="http://www.heh.be" target="_blank">Lien externe</a>
<a href="dossier/page.html#ancree">Lien interne ancré</a>
```

**Images**

```

```

**Éléments neutres (ni sémantique, ni mise en forme par défaut)**

```
<div>Bloc neutre</div>
<span>Texte neutre</span>
```

**Blocs sémantiques**

```
<header>En-tête</header>
<footer>Pied de page</footer>
<nav>Navigation</nav>
<article>Sujet à part entière</article>
<section>Section d'une page ou d'un article</section>
<aside>Compléments</aside>
<main>Bloc principal</main>
```

**Barres de progression et de mesure**

```
<meter min="0" max="100" value="52" low="50" high="60">52%</meter>
<progress max="100" value="52">52%</progress>
```

**Récapitulatif des sélecteurs CSS**

Sélecteurs	Syntaxe	Descriptions
<b>Sélecteur de type</b>	<code>x { }</code>	Toutes les balises <code>x</code>
<b>Sélecteur d'ID</b>	<code>#nom { }</code>	Toutes les balises dont l'ID est <code>nom</code>
<b>Sélecteur d'ID et de type</b>	<code>x#nom { }</code>	Toutes les balises de type <code>x</code> dont l'ID est <code>nom</code>
<b>Sélecteur de classe</b>	<code>.nom { }</code>	Toutes les balises dont la classe est <code>nom</code>
<b>Sélecteur de classe et de type</b>	<code>x.nom { }</code>	Toutes les balises <code>x</code> dont la classe est <code>nom</code>
<b>Sélecteur multiple</b>	<code>x, y { }</code>	Toutes les balises <code>x</code> et <code>y</code>
<b>Sélecteur descendant</b>	<code>x y { }</code>	Toutes les balises <code>y</code> contenue dans une balise <code>x</code> (descendants)
<b>Sélecteur d'enfant</b>	<code>x &gt; y { }</code>	Toutes les balises <code>y</code> directement contenues dans une balise <code>x</code> (enfants)
<b>Sélecteur universel</b>	<code>x * y { }</code>	Toutes les balises <code>y</code> contenues dans une balise quelconque, elle-même contenue dans une balise <code>x</code> (descendants non enfants)
<b>Sélecteur de frères adjacents</b>	<code>x + y { }</code>	Toutes les balises <code>y</code> s'ouvrant après la fermeture d'une balise <code>x</code>

**Responsive**

Réserver du CSS à certaines tailles de fenêtre `@media screen and (max-width: 760px) { }`

**Récapitulatif des propriétés CSS****Mise en forme des textes**

Couleur du texte	<code>color:#333 ;</code>
Police	<code>font-family:Trebuchet MS, Arial, sans-serif;</code>
Taille	<code>font-size:16px;</code>
Style	<code>font-style:italic; /* normal ; oblique */</code>
Petites capitales	<code>font-variant:small-caps; /* normal */</code>
Poids	<code>font-weight:bold; /* normal */</code>
Espacement entre lettres	<code>letter-spacing:0.2em;</code>
Interligne	<code>line-height:2em;</code>
Alignement	<code>text-align:center;</code> <code>/* left ; right ; justify */</code>
Soulignement	<code>text-decoration:none;</code> <code>/* underline ; overline */</code>
Indentation 1 <sup>re</sup> ligne	<code>text-indent:16px;</code>
Ombre (CSS3)	<code>text-shadow: 1px 1px 2px #999;</code>
Transformation de casse	<code>text-transform:lowercase;</code> <code>/* none ; uppercase ; capitalize */</code>
Alignement vertical d'éléments lignes côte à côte	<code>vertical-align:middle;</code> <code>/* sub ; super ; baseline ; top ; bottom ; text-top ; text-sup */</code>
Espacement des mots	<code>word-spacing:1em;</code>

**Polices exotiques importées**

Importer une police	<pre>@font-face {   font-family: "Starcraft";   src: url('fonts/Starcraft Normal.ttf'); } @font-face {   font-family: "Starcraft";   font-weight: bold;   src: url('fonts/Starcraft Bold.ttf'); }</pre>
Utiliser la police	<code>h1 {font-family: Starcraft, Arial, sans-serif; }</code>

## importée

## Coder une couleur en CSS

Différentes syntaxes de `#FF0000`

la couleur rouge `red`  
`#F00`  
`rgb(255,0,0)`  
`rgb(100%,0%,0%)`

Syntaxes CSS3 `rgba(255,0, 0, 1.0)`  
`hsl(0,100%,50%)`  
`hsla(0,100%,50%,1.0)`

## Marges et bordures

Marge intérieure `padding:16px;`  
 (remplissage) `padding:0 2em;`  
`padding:8px 0 16px 1em;`  
`padding-top:4px;`

Marge extérieure `margin:8px;`

Bordure `border:2px solid #333 ;`  
`border-width:8px;`  
`border-color:#00F;`  
`border-style:solid;`  
`/* none ; hidden ; dashed ; dotted ; double ;`  
`groove ; ridge ; inset ; outset */`

## Quelques autres propriétés

Opacité `opacity:0.5 ;`

Curseur `cursor: pointer ;`  
`/* crosshair, help, wait, alias, all-scroll, auto,`  
`cell, context-menu, col-resize, copy, default,`  
`grab, grabbing, help, move, no-drop, none, not-`  
`allowed, text, vertical-text, zoom-in, zoom-out,`  
`e-resize, ew-resize, n-resize, nesw-resize, ns-`  
`resize, nw-resize, nwse-resize, row-resize, s-`  
`resize, se-resize, sw-resize, w-resize */`

Style de liste `list-style:none ;`

Visibilité `visibility:hidden ; /* visible */`

## Centrer un élément

Centrer du texte `<div id= ;"main"><span>Texte</span></div>`  
`div { text-align :center ; }`  
`/* à appliquer à un ancêtre bloc */`

Centrer un bloc `<div id="main">...</div>`  
`#main { margin:0 auto ; width:960px; }`

**Arrière-plans**

<b>Couleur de fond</b>	<code>background-color:#eed;</code>
<b>Image de fond</b>	<code>background-image:url(chat.jpg);</code>
<b>Répétition de l'image</b>	<code>background-repeat:no-repeat ;</code> <code>/* repeat-x ; repeat-y ; repeat */</code>
<b>Défilement de l'image</b>	<code>background-attachment:fixed ;</code> <code>/* scroll */</code>
<b>Position de l'image</b>	<code>background-position:top right ;</code> <code>/* top/bottom/center left/right/center ;</code>
<b>Tout d'un coup</b>	<code>background: red url(image.gif) no-repeat fixed 25% 0%;</code>
<b>Taille de l'image (CSS3)</b>	<code>background-size:600px 400px ;</code> <code>/* cover ; contain */</code>
<b>Origine de l'image</b>	<code>background-origin: ... ;</code> <code>/* border-box ; padding-box ; content-box */</code>
<b>Couper l'image</b>	<code>background-clip: ... ;</code> <code>/* border-box ; padding-box ; content-box */</code>
<b>Images multiples</b>	<code>background: url(image1.png) no-repeat top left,</code> <code>url(image2.png) no-repeat top right,</code> <code>url(image3.png) repeat-x bottom left;</code>

**Positions**

<b>Modifier le rendu CSS</b>	<code>display:block ;</code> <code>/* block ; inline ; inline-block ; none */</code>
<b>Flottant</b>	<code>float:left ;</code> <code>/* left ; right ; none */</code>
<b>Espace latéral dégagé</b>	<code>clear:both ;</code> <code>/* left ; right ; none */</code>
<b>Positions</b>	<code>position:absolute ;</code> <code>/* static ; fixed ; relative ;</code>
<b>Éloignements</b>	<code>top:... ;</code> <code>bottom:... ;</code> <code>left:... ;</code> <code>right:... ;</code>
<b>Ordre de superposition</b>	<code>z-index:9 ;</code>
<b>Contenu dépassant</b>	<code>overflow:hidden ;</code> <code>/* visible ; scroll ; auto */</code>



# Projet : Portfolio OnePage Responsive

## 1. Sujet du projet

Le projet est un site web « One-Page » présentant votre CV et reprenant différents travaux de votre choix.

## 2. Obligations (le non-respect de ces obligations entraîne la cote de 0/20 pour le projet)

- **Poids** : le site ne doit pas dépasser les 10Mo.
- **Langages** : le site doit être codé en HTML5, en CSS et en JavaScript (+ bibliothèques). L'utilisation d'autres langages est proscrite.
- **Signature** : spécifiez votre nom en tant qu'auteur du site sous forme de méta-balise.
- **One-Page** : le site doit comporter 1 seul fichier html appelé **index.html**
- **Remise du projet** : à remettre au plus tard le **vendredi 26 mai 2017** via la plateforme Moodle sous forme de dossier compressé en **.zip** ou **.rar** uniquement.
- **Orthographe** : 5 fautes d'orthographe ou plus = 0/20 au projet

## 3. Critères d'évaluation

Critères	Points	Commentaires
<b>Fichiers</b> : structurez votre dossier ; trie les images dans des sous-dossiers ; ne remettez que des fichiers utiles ; surveillez le poids et le format des images ; choisissez minutieusement le nom des fichiers pour le référencement.		
<b>Contenu, apparence et ergonomie</b> : la quantité d'information, la qualité, le respect des règles d'ergonomie web et l'apparence de votre site.		
<b>Responsive</b> : choisissez et appliquez une des méthodes vues au cours afin que votre site s'adapte à toutes les résolutions existantes de 320px à 1366px.		
<b>Compatibilité</b> : vérifiez que votre site soit accessible dans IE8+, Chrome et Firefox.		
<b>HTML</b> : veillez à la lisibilité du code, à son indentation, au respect des normes du W3C et au référencement.		
<b>CSS</b> : veillez à l'ordre de vos déclarations et à leur répartition en sous-sections désignées par des commentaires (par exemple : <i>header, nav, content, marge, footer</i> ) ; appliquez au minimum une transition CSS3		
<b>JavaScript</b> : intégrez au minimum 3 animations/plugins jQuery différents (par exemple pour la navigation/scroll, un slider ou une galerie d'images avec effet lightbox...).		
<b>Total</b>		

## 4. Étapes de réalisation

Histoire de pas faire n'importe quoi, commencez par vous inspirer de **sites One-Page existants**, des curriculums de préférence. Le Web regorge de bonnes idées. N'oubliez pas que si vous réalisez correctement ce projet, il pourrait vous servir l'année prochaine lors de la recherche d'un stage ou plus tard lors de la recherche d'un emploi.

Recherchez des **conseils** en matière de rédaction de **CV** et de lettre de motivation. Faites simple, précis et efficace. Ne racontez pas votre vie. **Vendez-vous !**

Listez les **contenus** de votre site (par exemple : photo de profil, texte d'intro ou de présentation, CV, compétences, hobbies, travaux, etc. )

Pensez au **référencement** de votre site et à son **ergonomie**. Si besoin, relisez rapidement votre syllabus de 1<sup>re</sup>.

Griffonnez une **ébauche** de site sur papier pour avoir une idée de l'apparence avant de coder.

Commencez par la structure **HTML**, puis vérifiez la validité de votre code sur le Validator du W3C.

Attaquez ensuite la mise en page **CSS**, le **Responsive** et terminez par les animations **jQuery**.

Terminez par vérifier la **compatibilité** sur IE8+, Chrome et Firefox. **Compatible veut dire accessible, pas identique !**

Vérifiez une dernière fois **l'orthographe** et si besoin est, faites un copier/coller de tout votre site dans un correcteur orthographique quelconque.

## Sources

BEUZIT Patrick, aide-mémoire HTML 4 et CSS, éditions Eyrolles, 2002  
LAWSON Bruce & SHARP Remy, Introduction à HTML5, éditions Pearson, 2012  
MONCUR Michael, JavaScript, CampusPress, 2001  
VAN LACKER Luc, JavaScript – Introduction et notions fondamentales, éditions eni, 2008

Alsacrations : <http://css.alsacreations.com/>  
Comment ça marche : <http://www.commentcamarche.net/contents/>  
Développez.com : <http://www.developpez.com/>  
jQuery : <http://jquery.developpeur-web2.com/>  
Tout programmer : <http://www.toutprogrammer.com/>  
Validator W3C : <http://validator.w3.org/>  
W3Schools : <http://www.w3schools.com/>