COURS HTML EMBARQUÉ

Ce document est le support du cours “HTML embarqué” donné au Microclub en janvier et février 2018.

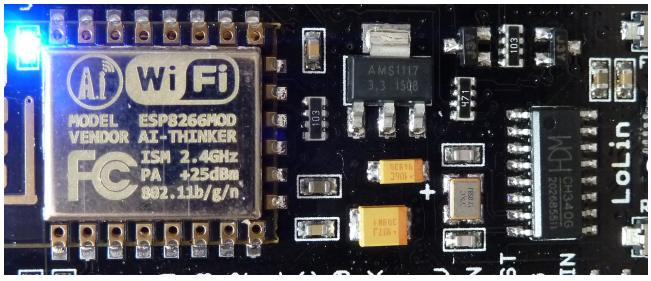
Le but de ce cours est de donner les bases qui permettront aux participants de créer des applications web simples et suffisamment légères pour être embarquées dans des objets connectés.

Nous aborderons les points suivants :

- HTML
- CSS
- JavaScript

Nous verrons lors du dernier cours comment appliquer pratiquement les point abordés en programmant une carte à microcontrôleur ESP8266.

Nicolas Jeanmonod



H T M L

Retour à l'accueil

LES AGENTS UTILISATEURS

Dans le cadre de ce cours, un *agent utilisateur*, ou *user agent* en anglais, est un logiciel qui peut traiter des données conformes aux normes HTML, XHTML, CSS, JavaScript et JSON. Cette liste n'est pas exhaustive et différents agents utilisateurs offrent des possibilités plus étendues ou au contraire plus restreintes. Pour la suite, nous nous focaliserons sur la partie du traitement des fichiers HTML, CSS et JavaScript.

Dans la grande majorité des cas, l'agent utilisateur sera un navigateur web, ou *web browser* en anglais, mais il est intéressant de noter que d'autres possibilités existent :

- Les appareils IoT (*Internet of Things* ou Internet des objets).
- Les navigateurs braille pour les personnes malvoyantes.
- Les logiciels de mise en page comme Prince XML.
- Les logiciels qui ne sont pas des navigateurs mais qui utilisent des données distantes. Par exemple les applications météo ou les horaires de transports publics.
- ...

Pour en savoir plus

- Agents utilisateurs (Wikipedia fr)

LA STRUCTURE MINIMALE D'UNE PAGE HTML

Voici la structure minimale pour qu'un document soit considéré comme un document HTML par tous les agents utilisateurs.

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

On voit dans cet exemple que le code HTML est composé de balises. Les balises absolument obligatoires pour qu'un document soit considéré comme un document HTML sont :

```
1. <!DOCTYPE html>
2. <html></html>
3. <head></head>
4. <meta charset=utf-8 />
5. <title></title>
6. <body></body>
```

Nous aurons besoin d'autres balises pour étoffer nos documents HTML, mais les 6 ci-dessus doivent toujours être présentes.

Sans entrer dans les détails, voici à quoi servent ces premiers éléments :

1. `<!DOCTYPE html>`
Définit que le document suit la norme HTML 5.

2. `<html></html>`
Tout le document (à l'exception du doctype) est entouré des balises `<html>`.

3. `<head></head>`
Le contenu de l'élément `<head>` sert à donner des informations aux agents utilisateurs. Ces informations ne sont donc pas visibles directement par l'utilisateur, à l'exception du contenu de l'élément `<title>` et du favicon (pas déclaré dans cet exemple). Il est possible de générer dynamiquement ce contenu avec JavaScript.

4. `<meta charset=utf-8 />`
Cet élément `<meta>` indique au navigateur quel est l'encodage du fichier. Sans cette information, les navigateurs font des choix par défaut qui sont souvent différents de l'UTF-8 et donc les caractères ne s'afficheront pas correctement. Pour des raisons de performance, il est conseillé de placer cette balise en premier dans l'élément `<head>`.

5. `<title></title>`
Le contenu de l'élément `<title>` sera affiché dans l'onglet de la page dans le navigateur.

6. `<body></body>`
Et enfin, le contenu visible par l'utilisateur de la page est placé dans l'élément `<body>`. Il est possible de générer dynamiquement ce contenu avec JavaScript.

LES ÉLÉMENTS ET LEURS BALISES

Structure des éléments HTML

Un élément HTML est formé de balises, ou *tag* en anglais. Par exemple :

```
<p></p>
```

Pour la balise ouvrante, la structure est :

- chevron ouvrant (`<`)
- le nom de la balise (dans l'exemple ci-dessus "`p`")
- chevron fermant (`>`)

Pour la balise fermante, la structure est :

- chevron ouvrant (`<`)
- barre oblique (`/`)
- le nom de la balise (dans l'exemple ci-dessus "`p`")
- chevron fermant (`>`)

Entre les balises, on peut inclure :

- rien : `<p></p>`
- du texte sans balises : `<p>du texte</p>`
- d'autres balises : `<p>du <i>texte</i></p>`
- d'autres balises imbriquées (*nested* en anglais) : `<p><b>du <i>texte</i></b></p>`

Éléments HTML vides

Certains éléments HTML sont toujours vides, c'est-à-dire qu'ils n'ont jamais de contenu entre leur balise ouvrante et leur balise fermante. On les appelle "éléments vides" (*empty elements* ou *void elements* en anglais). Dans ce cas, il est obligatoire d'utiliser une notation compacte avec la barre oblique avant le chevron fermant, que l'on nomme "balise autofermante".

Donc ceci est valide :

```
<br /> <!-- Line break = retour à la ligne -->
<img /> <!-- Image -->
<meta /> <!-- Meta-donnée -->
<link /> <!-- lien vers une feuille CSS -->
```

Alors que ceci n'est pas valide :

```
<br></br> <!-- Line break = retour à la ligne -->
<img></img> <!-- Image -->
<meta></meta> <!-- Meta-donnée -->
<link></link> <!-- lien vers une feuille CSS -->
```

> *Dans une balise autofermante, l'espace avant la barre oblique est optionnel.*

Voici une liste d'éléments vides : Éléments vides MDN, fr

Éléments HTML non vides

Si un élément peut avoir un contenu (*non-void HTML element* en anglais), mais que ce contenu est simplement vide dans un contexte donné, il n'est pas valide d'utiliser la notation compacte.

Donc ceci est valide :

```
<p></p> <!-- Paragraphe -->
<li></li> <!-- Élément de liste -->
<h1></h1> <!-- Titre de niveau 1 -->
```

Alors que ceci n'est pas valide :

```
<p /> <!-- Paragraphe -->
<li /> <!-- Élément de liste -->
<h1 /> <!-- Titre de niveau 1 -->
```

> *La grande majorité des éléments HTML sont non vides.*

Cas particulier de l'élément `<script>`

L'élément `script` est un peu spécial parce qu'il peut être utilisée de deux façons différentes :

- Pour appeler un script externe au fichier HTML.
- Pour définir un script interne au fichier HTML.

Dans le cas de l'appel du script externe, cet élément est toujours vide. Il serait donc légitime de penser qu'on peut utiliser la balise autofermante. Mais comme cet élément peut avoir un contenu dans sa deuxième utilisation, on doit toujours utiliser la notation avec une balise ouvrante et une balise fermante.

```
<!-- Lien vers un fichier JavaScript (JS) externe. -->
<script src="scripts.js"></script>

<!--
    Définition d'un script JavaScript (JS) interne.


  (exemple ci-dessus), même si l'élément est vide.
-->
<script>
    console.log("Hello Microclub !")
</script>
```

Notons au passage que les feuilles de style CSS peuvent aussi être internes ou externes au fichier HTML. Mais dans ce cas, les balises utilisées sont différentes :

```
<!--
    Lien vers une feuille de style CSS externe.
    Comme la balise <link> est toujours vide, elle doit être autofermée.
    -->
<link rel="stylesheet" href="style.css" />
```

```
<!-- Définition d'une feuille de style CSS interne. -->
<style rel="stylesheet">
    /* commentaire CSS */
</style>
```

Pour résumer :

CSS externe : `<link />`
CSS interne : `<style></style>`
JS externe : `<script></script>`
JS interne : `<script></script>`

Imbrication d'éléments

Lorsque les éléments sont imbriqués (*nested* en anglais), il est important de fermer les balises "enfants" avant les balises "parents".

Ceci est correct :

```
<p>
    <strong>...</strong>
</p>
```

Ceci ne l'est pas :

```
<p>
    <strong>...</p>
</strong>
```

Quelques éléments courants

Voici une liste non exhaustive d'éléments courants :

```
<p>Ceci est un paragraphe.
Les retours à la ligne doivent y
être explicitement indiqués avec la balise <br />
&lt;br />
La balise "p" est la plus utilisée pour afficher du texte.
</p>

<p><strong>Texte en gras</strong></p>

<p><em>Texte en italique</em></p>

<p><strong>Texte en gras <em>avec une partie italique</em></strong></p>

<pre>Ceci est un texte préformaté
     dans lequel les passages à la ligne
```

```
     et les espaces
     seront respectés</pre>

<p>Le <span>est</span> utilisé pour formater différemment une partie du texte.</p>

<div>Le div sert à regrouper des balises.
    <p>...</p>
    <p>...</p>
    <div></div>
</div>

<img src="https://ouilogique.com/images/site-logo.png" alt="image de démo" />
```

Liste des balises HTML

La norme HTML définit environ 130 balises différentes que vous pouvez découvrir en suivant les liens ci-dessous.

Éléments vs balises

Comme nous l'avons vu ci-dessus, un élément HTML est composé de balises et d'un contenu. Dans le langage courant, on utilise souvent indifféremment les termes *éléments* et *balises*, même si à proprement parler il s'agit de deux choses distinctes.

Pour en savoir plus

- Les balises HTML et leur rôle (MDN fr)
- Référence des éléments HTML (MDN fr)
- HTML Element Reference (W3Schools en)

LES ATTRIBUTS

Toutes les balises acceptent des attributs, certains étant obligatoires d'autres optionnels.

Par exemple, la balise `<img>` qui sert à insérer une image a deux attributs obligatoires : `src` et `alt`. À noter que l'attribut `alt` est souvent omis dans les pages web que vous rencontrerez. Il est pourtant fortement conseillé, car il s'agit du texte qui remplacera l'image si celle-ci ne peut pas être affichée. Cet attribut est aussi utilisé par les systèmes de lecture pour les malvoyants.

```
<img src="https://ouilogique.com/images/site-logo.png" alt="image de démo" />
```

Voir les attributs possibles de la balise `<img />` sur le site MDN.

Les attributs peuvent être mis à ligne pour faciliter la lecture :

```
<img
    src="https://ouilogique.com/images/site-logo.png"
    alt="image de démo" />
```

Les attributs personnalisés

HTML5 permet à l'utilisateur de définir ses propres attributs ce qui peut être fort utile lorsqu'on désire rendre le document interactif avec JavaScript. Ces attributs personnalisés s'appellent "attributs de données" et doivent obligatoirement commencer par le préfix `data-`.

```
<article
  id="voitureelectrique"
  data-columns="3"
  data-index-number="12314"
  data-parent="voitures">
...
</article>
```

Voir les attributs de données sur MDN

LE DOCTYPE

Le doctype est une chaine de caractère présente au début du fichier et qui définit explicitement la version de la norme HTML utilisée dans le document. Le mot *doctype* est un mot-valise tiré de la locution anglaise *Document Type Declaration*. Le seul doctype que nous utiliserons dans le cadre de ce cours est le doctype HTML 5 qui se déclare de la manière suivante :

```
<!DOCTYPE html>
```

Si on ne spécifie pas de doctype, alors les agents utilisateurs utiliseront le *mode quirks*, c'est-à-dire que le moteur de disposition émule le comportement non standard de Netscape Navigator 4 et d'Internet Explorer 5. Ce mode permet de prendre en charge les sites web rédigés avant l'adoption généralisée des standards web.

Si on spécifie un doctype, alors les agents utilisateurs utilisent le mode standard total ou éventuellement le mode quasi standard (qui a priori n'existe plus).

Si vous désirez plus d'informations sur ces différents modes, vous pouvez vous référer aux liens ci-dessous. Je vous conseille de toute façon de ne pas jouer avec le feu et de toujours spécifier le doctype HTML 5 (`<!DOCTYPE html>`).

Pour déterminer si un agent utilisateur est en mode standard ou au contraire en mode quirks, vous pouvez utiliser le JavaScript suivant :

```
console.log(document.compatMode === "CSS1Compat" ? "standard" : "quirks");
```

Historiquement, de nombreux doctype ont eu cours et il est fort probable que vous en rencontriez d'autres.

Le doctype est un élément à part dans la grammaire HTML dans le sens où sa balise ouvrante ne doit jamais être fermée.

Pour en savoir plus

- Recommended list of Doctype declarations (W3C en)
- Document type declaration (Wikipedia en)
- Doctype (Wikipedia fr)
- Mode quirks de Mozilla (MDN fr)
- Mode presque standard de Gecko (MDN fr)
- Fix Your Site With the Right DOCTYPE!

L'ESPACE DE NOM

Certains validateurs comme celui de l'éditeur *Oxygen XML Editor* imposent que l'espace de nom soit spécifié et ceci se fait dans l'attribut `xmlns` de la balise `<html>` ouvrante :

```
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>
```

LES ÉDITEURS DE CODE HTML

Voici une liste non exhaustive d'éditeurs de code HTML :

- Brackets (que nous allons utiliser pour ce cours)
- Sublime Text 3 (que j'utilise au quotidien)
- Atom
- Oxygen XML Editor
- Visual Studio Code
- Notepad++
- BBEdit
- Gedit
- Nano
- ...

LA VALIDATION

À ce stade, nous pouvons commencer à vérifier que ce que nous faisons est valide avec le validateur du World Wide Web Consortium (W3C) W3C Markup Validation Service.

LES ENTITÉS

Les entités servent à référencer les caractères par un code qui peut être textuel, décimal ou hexadécimal. Ceci est particulièrement utile dans les cas suivants :

- Le caractère est aussi utilisé dans la grammaire HTML comme les signes <, > et &.
- Le caractère peut être confondu avec un autre ayant un glyphe visuellement identique, comme l'espace insécable qui est visuellement identique à l'espace simple et que l'on représentera par donc par ` ` ou le trait d'union insécable, identique au trait d'union normal et que l'on représentera par `&#8209;`.
- Le type d'encodage du fichier texte ne permet pas de représenter le caractère. Par exemple si le fichier est encodé en `windows-1257`, le caractère "ç" ne sera pas utilisable directement, mais pourra quand même être représenté avec l'entité `&ccedil;`.
- Le caractère ne s'affiche pas correctement dans votre éditeur, comme l'émoticône "visage souriant" (`&#9786;` = ☺).
- Le caractère est par nature invisible, comme l'espace sans chasse (`&#8203;`).

> *Pour tout nouveau projet, il est important de s'assurer que tous vos fichiers sont encodés en `utf-8` qui est une norme quasi universelle aujourd'hui en occident. Le premier avantage est que vos fichiers seront lisibles par la grande majorité des agents utilisateurs et le deuxième avantage est que vous ne serez pas contraint d'utiliser plus d'entités que nécessaire.*

Quelques entités

Caractère	Glyphe	Entité textuelle	Entité décimale	Entité hexadécimale
Signe inférieur à	<	`<`	`&#60;`	`&#x3C;`
Signe supérieur à	>	`>`	`&#62;`	`&#x3E;`
Esperluette	&	`&`	`&#38;`	`&#x26;`
Espace insécable		` `	` `	` `
Trait d'union insécable	-		`&#8209;`	`&#x2011;`
Émoticône "visage souriant"	☺		`&#9786;`	`&#x263A;`
Espace sans chasse			`&#8203;`	`&#x200B;`

Vous trouverez une liste exhaustive d'entités sur unicode-table.com.

HTML VS XHTML

La norme HTML 5 autorise deux syntaxes :

- La syntaxe HTML
- La syntaxe XHTML

La différence la plus notable est que la syntaxe HTML autorise que certaines balises ne soient pas fermées :

en HTML, ceci est valide :

```
<meta charset="utf-8">
<p>...
<ul>
    <li>...
    <li>...
</ul>
```

alors qu'en XHTML, toutes les balises doivent être fermées :

```
<meta charset="utf-8" />
<p>...</p>
<ul>
    <li>...</li>
    <li>...</li>
</ul>
```

> *La syntaxe XHTML a l'immense avantage d'être logique et facile à appliquer lorsque l'on génère le code à la main. De plus elle permet d'utiliser les outils prévus pour les fichiers XML, comme les langages XPath, XSLT et XQuery.*
>
> *La syntaxe HTML permet quand à elle d'économiser quelques octets puisque le nombre de balises nécessaires est moindre. Certaines personnes, dont je ne fais pas partie, la trouvent aussi plus facile à lire puisque moins verbeuse.*
>
> *Le validateur du W3C accepte le mélange des deux syntaxes dans un même document et de ce que je peux voir sur le web, la syntaxe HTML a le vent en poupe. Cependant,* ***mon conseil est de privilégier ~~systématiquement~~ la syntaxe XHTML.*** *(Je découvre quelques mois plus tard que Google suggère de privilégier la syntaxe HTML.)*

Les différences principales entre HTML et XHTML

Structure du document

- Le doctype `<!DOCTYPE html>` est obligatoire.
- L'attribut `xmlns="http://www.w3.org/1999/xhtml"` est obligatoire.
- Les balises `<html>`, `<head>`, `<title>`, et `<body>` sont obligatoires.

Balises XHTML

- Les balises doivent être imbriquées correctement.
- Les balises doivent être systématiquement fermées ou autofermées.
- Les balises doivent être écrites en minuscules.
- La balise racine `<html>` doit être unique.

Attributs XHTML

- Les attributs doivent être écrits en minuscules.
- Les valeurs des attributs doivent être entourées de guillemets simples (`'`) ou doubles (`"`).
- La minimisation des attributs est interdite.
  `FAUX  ⇒ <input checked />`
  `JUSTE ⇒ <input checked="checked" />`

> *Note : Je préfère souvent les guillemets doubles (`"`) aux guillemets simples (`'`), parce que l'apostrophe sur un clavier standard est aussi un guillemet simple et que ça peut rendre les recherches fastidieuses, particulièrement quand on veut appliquer les règles de typographie soignées qui imposent d'utiliser l'apostrophe typographique (`’`) au lieu de l'apostrophe droite (`'`).*

Pour en savoir plus

- HTML and XHTML (W3Schools en)
- Apostrophe et « impostrophe »
- Google HTML/CSS Style Guide

LE TRAITEMENT DES BLANCS

> *On appelle "blanc" ou whitespace en anglais, un caractère qui n'a pas de représentation graphique. Les blancs les plus usuels sont le retour à la ligne et l'espace (qui est un mot féminin en typographie).*

En HTML et en CSS, il y a 5 façons possibles de traiter les blancs :

- **normal**
  Les séries de blancs sont regroupées, les caractères de saut de ligne sont gérés comme les autres blancs. Les passages à la ligne sont faits naturellement pour remplir les boîtes.

- **nowrap**
  Les blancs sont regroupés comme avec normal mais les passages à la ligne automatiques sont supprimés.

- **pre**
  Les séries de blancs sont conservées telles quelles. Les sauts de ligne ont uniquement lieu avec les caractères de saut de ligne et avec les éléments `<br />`.

- **pre-wrap**
  Les séries de blancs sont conservées telles quelles. Les sauts de ligne ont lieu avec les caractères de saut de ligne, avec `<br />` et on a des passages à la ligne automatiques.

- **pre-line**
  Les séries de blancs sont regroupées, les sauts de lignes ont lieu avec les caractères de saut de ligne, les éléments `<br />` et on a des passages à la ligne automatiques.

Exemple de blanc *normal* avec la balise `<p>`

- Les espaces successives ne comptent que pour une espace.
- Si on veut tout de même afficher plusieurs espaces successives, il faut utiliser l'espace insécable (` `).
- Un ou plusieurs retours à ligne sont traités comme une seule espace.
- Si on veut un retour à ligne, on doit utiliser la balise `<br />` (*line break*).

Valeurs par défaut

- **normal**
  La grande majorité des balises HTML ont leur attribut CSS `white-space` défini par défaut à la valeur `normal`.

- **pre**
  `<option>`, `<pre>`, `<select>`

- **pre-wrap**
  `<textarea>`

Modification de la valeur par défaut

On peut modifier la valeur de l'attribut `white-space` par défaut avec l'instruction CSS suivante :

```
p.pre-line { white-space: pre-line; }
```

Pour en savoir plus

- white-space (MDN fr)
- Un espace ou une espace ?

LES COMMENTAIRES

Si l'on veut commenter une portion de code HTML, il faut utiliser la notation suivante :

```
<!-- Commentaire -->
```

Par contre, il faut utiliser les marques de commentaire natives des langages que l'on intègre :

Exemple avec du code CSS intégré dans le code HTML :

```
<style rel="stylesheet">
    /* commentaire CSS */
</style>
```

Exemple avec du code JavaScript intégré dans le code HTML :

```
<script>
    /* commentaire JS */
    // Commentaire JS
</script>
```

C S S

Retour à l'accueil

INTRODUCTION

Nous avons vu que le langage HTML permet de définir la structure du contenu d'un document. Nous allons maintenant voir comment mettre en forme l'apparence de ce contenu grâce au langage CSS.

CSS signifie *Cascading Style Sheets*, c'est à dire *Feuilles de style en cascade*. CSS fait appel à des sélecteurs qui permettent d'appliquer les styles aux différents éléments HTML.

Note

Il n'y a pas que les documents HTML qui peuvent profiter de la mise en forme avec CSS, les documents SVG (*Scalable Vector Graphics* ou graphique vectoriel adaptable, en français) le peuvent aussi.

Pour en savoir plus

- CSS sur MDN

BÉNÉFICES DE LA TECHNOLOGIE CSS

Une des grandes forces de la technologie CSS est d'offrir la possibilité de mettre en forme un document de façons complètement différentes en adaptant uniquement les feuilles de style, mais sans modifier le document HTML source.

Ce cours lui-même est un exemple de cette possibilité offerte par CSS. Il y a plusieurs scénarios de transformation possibles et chacun à son CSS associé :

Exemple avec ce cours

Prérequis

- Rédaction des documents au format Markdown.

Scénario 1 : Génération avec GitHub

- Lors de la mise en ligne, GitHub convertit automatiquement les fichiers Markdown en HTML et affiche le résultat sur leur site en utilisant leur propre CSS.

Scénario 2 : Génération avec Pandoc

- J'utilise Pandoc pour convertir les fichiers Markdown en HTML et j'inclus dans le fichier HTML la feuille de style CSS qui permet de lire ce fichier dans un navigateur.

Scénario 3 : Génération avec Prince

- J'utilise Prince pour convertir le fichier HTML du scénario 2 en format PDF. La feuille de style est également la même, mais elle contient des instructions qui permettent de différencier le rendu à l'écran et le rendu au format PDF.

Conclusion

On voit que les trois scénarios utilisent des CSS différents et qu'un fichier source peut générer différents fichiers cibles.

EXEMPLE AVEC *CSS ZEN GARDEN*

Un exemple spectaculaire de la puissance de la technologie CSS est le site *CSS Zen Garden*. Le défi proposé par ce site est que tous les participants mettent en forme le même fichier HTML et le résultat visuel doit être aussi original que possible. Je vous laisse juger de l'inventivité des participants ainsi que de la puissance de la technologie CSS.

POSITION DU CSS

Les instructions CSS peuvent être définies dans 3 endroits différents :

1. Dans un ou plusieurs fichiers externes.
2. À l'intérieur du fichier HTML, dans la balise `<style>` qui doit se trouver elle-même dans la balise `<head>`.
3. Dans l'attribut `style` d'un élément HTML.

CSS externe au fichier HTML

L'avantage du fichier CSS externe est qu'il permet de réunir toutes les informations de style en un seul fichier qui pourra être utilisé par toutes les pages HTML d'un site, aussi grand soit-il. Ceci permet de minimiser la bande passante et d'accélérer le rendu puisque les agents utilisateurs peuvent garder la feuille de style en cache.

Avec des fichiers CSS externes, il est également plus facile de gérer des scénarios de transformation différents.

Voici un exemple de fichier CSS (style.css) :

```
p {
    color: firebrick;
}
```

Pour faire appel à ce fichier, il faut inclure la balise `<link>` dans le document HTML, généralement à l'intérieur de la balise `<head>`, mais la norme HTML 5 permet de positionner cette balise dans l'élément `<body>` également.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>CSS</title>
    <link rel="stylesheet" href="style.css" />
</head>
<body>
    <p>Hello Microclub !</p>
</body>
</html>
```

CSS interne au fichier HTML, dans la balise `<style>`

Si on désire ne servir qu'un seul fichier qui contient à la fois les instructions HTML et CSS, comme c'est souvent le cas dans le monde de l'IoT, il est possible de placer les instructions CSS dans une ou plusieurs balises `<style>` qui doivent se trouver dans la balise `<head>` du fichier HTML.

On peut aussi utiliser cette méthode pour créer des exceptions pour une page donnée dans un site web contenant plusieurs pages.

C'est aussi un moyen de forcer les agents utilisateurs à ne pas réutiliser les informations de leur cache.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>CSS</title>
    <style>
    p {
    color: aqua;
    }
    </style>
</head>
<body>
    <p>Hello Microclub !</p>
</body>
</html>
```

CSS interne au fichier HTML, dans l'attribut `style` d'un élément HTML

Cette méthode est analogue à celle que l'on utiliserait dans un logiciel de traitement de texte, c'est-à-dire que l'on applique le style directement à l'élément concerné.

Elle doit être utilisée le moins souvent possible, car elle augmente considérablement la taille des fichiers et diminue la lisibilité. Google pénalise aussi cette pratique lors de l'indexation si on en croit leur outil PageSpeed Insights.

Elle a également le désavantage d'empêcher la gestion de scénarios de transformation.

Cela dit, c'est un bon moyen de créer une exception pour un élément donné.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>CSS</title>
</head>
<body>
    <p style="color: blueviolet">Hello Microclub !</p>
</body>
</html>
```

LA SYNTAXE

Pour apprendre les bases de la syntaxe CSS, nous allons utiliser l'exemple ci-dessous (adapté de l'exemple sur MDN).

fichier `syntaxe.css`

```
body {
    font: 1em/150% "Helvetica Neue", Arial, monospace;
    padding: 1em;
    margin: 20px auto;
    max-width: 33em;
}

@media (min-width: 768px) {
    body {
    border: 1px solid chocolate;
    }
}

h1 {
    font-size: 1.5em;
}

div p, #id:first-line {
    background-color: antiquewhite;
    color: cadetblue;
}

div p {
    margin: 0;
    padding: 1em;
}

div p + p {
    padding-top: 0;
}
```

fichier `syntaxe.html`

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Syntaxe CSS</title>
    <link rel="stylesheet" href="syntaxe.css" />
</head>
<body>
    <h1>Un exemple simple</h1>

    <p id="id">Lorem ipsum dolor sit amet, consectetur adipiscing elit.
       Nulla eget sapien volutpat, blandit tortor sit amet, consequat
       sem. Mauris suscipit nunc eu mi lobortis, in porttitor nulla
       tempor. Suspendisse ante erat, eleifend auctor dictum in,
       viverra eu elit.</p>

    <div>
      <p>Nullam sit amet augue consequat, tristique enim non, varius
         orci. Vivamus bibendum elit turpis, sit amet fringilla neque
         mollis sed. Donec semper, nibh molestie maximus pretium, tellus
         nibh molestie leo, vitae vulputate ante turpis a eros.</p>

      <p>Vestibulum pharetra metus id quam dignissim, ac maximus libero
         efficitur. Nullam hendrerit diam nisl. Nullam feugiat semper
         ipsum a pharetra. Ut posuere varius consectetur. Sed purus nunc,
         fringilla laoreet risus vitae, laoreet dapibus diam.</p>
    </div>
</body>
</html>
```

Anatomie d'une règle CSS

Une règle CSS se présente sous la forme suivante :

```
sélecteur1, sélecteur2 {
    propriété1: valeur1;
    propriété2: valeur2;
}
```

Voici un exemple réel de règle :

```
div p, #id:first-line {
    background-color: antiquewhite;
    color: cadetblue;
}
```

On voit qu'une règle est composée de la manière suivante :

- Un ou plusieurs **sélecteurs** séparés par des virgules, ici `div p` et `#id:first-line`.
- Un bloc délimité par des accolades `{}`.
- Un **bloc** peut être vide ou contenir une ou plusieurs **déclarations**, ici `background-color: antiquewhite;` et `color: cadetblue;`.
- Les blocs peuvent être imbriqués.

- Chaque déclaration est constituée de **propriétés** et de **valeurs** séparées du caractère deux points (`:`).
- Les déclarations sont terminées par un point-virgule.
- Le point-virgule est optionnel (mais recommandé) pour la dernière déclaration d'un bloc.
- Les déclarations peuvent être mises sur une ou plusieurs lignes.

> ***Les instructions CSS sont sensibles à la casse (majuscule ≠ minuscule).***
> *En CSS, il n'est pas possible de définir des variables ou des constantes, ce qui est très problématique pour gérer des projets complexes. Dans ce cas, on utilisera un langage de génération de feuilles de style comme Sass.*
> *Si une règle est invalide, elle est ignorée.*

Commentaires

Il est possible de commenter le code CSS de la manière suivante :

```
body {
    font: 1em/150% "Helvetica Neue", Arial, monospace;
    padding: 1em;
    /*
    Ceci est un commentaire.
    Les instructions ci-dessous
    ne seront pas exécutées
    car elles sont commentées.
    */
    /*
    margin: 20px auto;
    max-width: 33em;
    */
}
```

> *CSS ne permet pas de commenter ligne par ligne, comme il est possible de le faire en JavaScript avec les symboles `//`.*

> *On ne peut pas commenter une portion de code qui contient elle-même des commentaires. Ceci est très contraignant et il convient donc d'être très prudent quand on commente une grande portion de code. Si des commentaires étaient déjà présents, le code ne sera plus valide !*

Validation

Pour s'assurer qu'un code CSS est valide, on peut utiliser le validateur CSS du W3C.

Pour en savoir plus

- Les déclarations CSS MDN
- Les blocs CSS MDN
- Les règles CSS MDN

Les instructions CSS

Les instructions CSS MDN

LES SÉLECTEURS

En CSS, les sélecteurs sont utilisés afin de cibler une partie spécifique d'une page web à mettre en forme. Afin de pouvoir être précis, CSS est très riche en sélecteurs et une grande partie de sa flexibilité dépend de ceux-ci.

- Les sélecteurs simples
    - Les sélecteurs de type (type selectors)
    - Les sélecteurs de classe
    - Les sélecteurs d'identifiant
    - Le sélecteur universel
- Les sélecteurs d'attribut
    - Définition et valeur des sélecteurs d'attribut
    - Les sélecteurs d'attribut utilisant un filtre sur les fragments de chaînes
- Les pseudo-classes
- Les pseudo-éléments
- Les combinateurs

Pour découvrir les sélecteurs, rendez-vous sur la page des sélecteurs de MDN.

LA CASCADE

Comme son nom l'indique, CSS agit en cascade, ce qui signifie que les définitions de style sont lues les unes après les autres et que si deux règles sont identiques, c'est la dernière qui sera appliquée.

Dans l'exemple ci-dessous, la couleur du texte à l'intérieur des balises `<p>` est définie 3 fois. Pour être exhaustif, il faut aussi mentionner que les agents utilisateurs appliquent eux aussi des styles par défaut et qu'ils peuvent permettre aux utilisateurs de créer leurs propres styles. Donc, l'agent utilisateur lira les 5 sources suivantes et dans cet ordre :

1. Style par défaut de l'agent utilisateur ⇒ *black*
2. Utilisateur de l'agent utilisateur ⇒ *not set*
3. Fichier externe (style.css) ⇒ *firebrick*
4. Balise `<style>` ⇒ *aqua*
5. Attribut `style` de l'élément `<p>` ⇒ *blueviolet*

Les règles de cascade imposent que, si des règles sont en concurrence, ce soit la dernière qui est appliquée. Donc dans l'exemple ci-dessous, la couleur du texte sera *blueviolet*.

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

```
    <!-- Lorsque l'agent utilisateur arrive ici,
    il a déjà chargé ses styles par défaut ainsi
    que les styles personnels de l'utilisateur. -->

    <!-- Ici l'agent utilisateur charge le fichier
    style.css -->
    <link rel="stylesheet" href="style.css" />

    <!-- Ici l'agent utilisateur charge les styles
    déclarés dans la balise "style" -->
    <style>
        p {
            color: aqua;
        }
    </style>
</head>
<body>
    <!-- Ici l'agent utilisateur applique le style
    indiqué dans l'attribut "style" -->
    <p style="color: blueviolet">Hello Microclub !</p>
</body>
</html>
```

LA PONDÉRATION

Les règles de la cascade s'appliquent pour prioriser les différentes sources. Que se passe-t-il si, pour une même source, plusieurs règles concernent le même élément ? Dans ce cas, pour prioriser les règles, on prendra en compte le poids du sélecteur. Le poids d'un sélecteur est calculé en fonction de sa spécificité :

1. **Niveau 1 (spécificité élevée)** : Sélecteur d'identifiant
2. **Niveau 2 (spécificité moyenne)** : Sélecteur de classe, de pseudo-classe et d'attribut
3. **Niveau 3 (spécificité faible)** : Sélecteur de type et de pseudo-element

Pour déterminer si une règle s'applique plutôt qu'une autre on regarde :

- Celle qui possède le plus de sélecteurs de niveau 1.
- Si le nombre de sélecteurs de niveau 1 est le même : celle qui possède le plus de sélecteurs de niveau 2.
- Si le nombre de sélecteurs de niveau 2 est le même : celle qui possède le plus de sélecteurs de niveau 3.
- Enfin, si les sélecteurs ont le même poids, ce sera l'ordre des règles dans le fichier source qui importera : la règle la plus basse dans le fichier l'emportera sur une règle déclarée avant.

Prenons ce fragment de HTML par exemple :

```
<p id="cookie" class="crispy">Ce cookie est <span>délicieux !</span></p>
```

Cette feuille de style illustre ce qui se passe lorsqu'un sélecteur de niveau 1 entre en conflit avec d'autres sélecteurs :

```
/* Ce sélecteur est composé d'un sélecteur de niveau 1
   Poids : 1 | 0 | 0 */
#cookie {
  color: green;
}
/* Ce sélecteur n'a aucun sélecteur de niveau 1
   et 2 sélecteur de niveau 2
   Poids : 0 | 2 | 0 */
[id=cookie].crispy {
  color: red;
}

/* 1|0|0 > 0|2|0 : La première règle s'applique et le texte
   est vert. Un seul sélecteur de niveau 1 sera toujours
   prioritaire par rapport à X sélecteurs de niveau 2. */
```

Pour en savoir plus

- Pondération MDN

L'HÉRITAGE

Héritage MDN

LES *MEDIA QUERIES*

Les *media queries*, ou requêtes media en français, sont des instructions CSS qui permettent d'appliquer des règles CSS différentes en fonction de l'appareil utilisé et ceci sans modification du code HTML. Elles sont à la base de ce qu'on appelle le *responsive design* ou design adaptatif en français et c'est ce qui permet à un site web de s'afficher lisiblement à la fois sur un écran de bureau 27" et sur l'écran d'un téléphone.

On peut spécifier le média cible à deux endroits différents :

Dans l'attribut `media` de la balise `<link>` qui appelle le code CSS :

```
<link rel="stylesheet" media="screen" href="style-screen.css" />
<link rel="stylesheet" media="print" href="style-print.css" />
```

Ou dans le CSS lui-même :

```
body
{
  background-color: lime;
}

@media (min-width: 700px)
{
  body
  {
```

```
    background-color: red;
  }
}

@media (min-width: 700px) and (orientation: landscape)
{
  body
  {
    background-color: lime;
  }
}
```

Pour en savoir plus

- Les medias queries MDN

LES COULEURS

Les couleurs peuvent être définies de plusieurs façons différentes. Tous les exemples ci-dessous définissent la même couleur firebrick.

```
/* Par nom : */
body { background-color: firebrick; }

/* Par code hexadecimal RGB : */
body { background-color: #B22222; }

/* Par code decimal RGB : */
body { background-color: rgb(178, 34, 34); }

/* Par code decimal RGB avec gestion de la transparence : */
body { background-color: rgba(178, 34, 34, 0.5); }

/* Par code decimal HSL : */
body { background-color: hsl(0, 68%, 42%); }

/* Par code decimal HSL avec gestion de la transparence : */
body { background-color: hsla(0, 68%, 42%, 0.5); }
```

Notation courte

Il existe aussi une notation courte sur 3 chiffres au lieu de 6. Par exemple, `#B22` est équivalent à `#BB2222`, ce qui est très proche de `#B22222` de notre exemple ci-dessus.

```
body { background-color: #B22; }
```

Nombre de couleurs

La notation standard sur 6 chiffres permet d'afficher $16^6 = 256^3 = 16'777'216$ de couleurs, alors que la notation sur 3 chiffres n'en offre que $16^3 = 4'096$.

Pour en savoir plus

- HTML Color Picker W3 Schools
- Couleurs CSS MDN

FRAMEWORKS

Mettre en forme des pages HTML peut vite devenir une tâche complexe, particulièrement quand on veut l'afficher sur des écrans de tailles très différentes. Pour cela, il est fortement conseillé de ne pas réinventer la roue et d'utiliser un *framework*, c'est-à-dire une collection d'outils prête à l'emploi.

Dans le monde du CSS, le *framework* Bootstrap est très populaire en ce moment. Il est basé sur un système de grille à 12 colonnes qui sont utilisées pour l'alignement des éléments visuels.

Un désavantage de Bootstrap est que le fichier CSS de base (bootstrap.min.css) pèse 118 ko, ce qui est relativement conséquent quand on veut travailler avec un ESP8266. Mais j'ai essayé et ça fonctionne. Si la topologie du projet le permet, il est de toute façon préférable de charger Bootstrap depuis un CDN et pas depuis le site web ou l'appareil IoT de l'application en utilisant le code suivant :

```
  K68vbdEjh4u" crossorigin="anonymous" />
```

Voici un canevas pour bien commencer avec Bootstrap. Il est assez complet, donc dans la plupart des cas, il conviendra d'enlever les éléments inutilisés.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="utf-8" />
  <title></title>
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />


  K68vbdEjh4u" crossorigin="anonymous" />


  Sp" crossorigin="anonymous" />
  <style type="text/css">

  css?family=Source+Sans+Pro:200,200i,400');
    body {
      font-family: 'Source Sans Pro', monospace;
      font-size: 14pt;
      margin: 2em;
      background-color: #f2f2f2;
    }
  </style>
```

```
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-xs-offset-0">
        <h1>Bootstrap starter template</h1>
        <p></p>
      </div>
    </div>
  </div>


  crossorigin="anonymous"></script>


  crossorigin="anonymous"></script>


  ie10-viewport-bug-workaround.js"></script>
</body>
</html>
```

Il existe d'autres *frameworks*, comme Foundation qui est aussi très populaire.

Pour en savoir plus

- Site officiel de Bootstrap, en
- Site officiel de Foundation, en

POUR LA SUITE

Avec cette introduction sur la technologie CSS, vous avez maintenant une base pour comprendre comment mettre en forme une page HTML simple.

Si vous désirez approfondir vos connaissances, vous pouvez lire les articles sur la technologie CSS sur le site MDN.

Pour vous aider lors de l'écriture de vos feuilles de style, vous trouverez une liste exhaustive des mots-clés de la syntaxe CSS ici : Référence CSS MDN.

Le site du W3C contient lui aussi beaucoup d'informations utiles : CSS sur le site du W3C.

JAVASCRIPT

Retour à l'accueil

INTRODUCTION

Nous avons vu que le langage HTML permet de définir la structure du contenu d'un document et que le langage CSS permet de mettre en forme ce contenu. Nous allons maintenant voir comment utiliser le langage JavaScript pour rendre ce contenu interactif.

L'ORIGINE DU NOM JAVASCRIPT

JavaScript aurait dû s'appeler LiveScript, mais a été renommé par une décision marketing dans le but de capitaliser sur la popularité du langage Java de Sun Microsystems, malgré le fait qu'ils n'aient que très peu en commun. Cela a toujours été une grande source de confusion.

JavaScript est normalisé par l'Ecma International (curieusement en Europe) sous la norme ECMA-262 et sous le nom de langage ECMAScript. Les noms ECMAScript et JavaScript sont donc souvent considérés comme interchangeables. Cependant, il faut noter que d'autres langages suivent aussi la norme ECMA-262, comme Flash 5 ActionScript d'Adobe ou JScript de Microsoft.

Pour en savoir plus

- Differences from ECMA-262 and JavaScript

UTILISATION DE JAVASCRIPT

JavaScript est principalement utilisé dans les navigateurs web, mais on le trouve aussi dans d'autres environnements tels que Node.js, Apache CouchDB voire Adobe Acrobat.

JavaScript permet d'automatiser certaines tâches qui rendront les pages web interactives. Dans le contexte de ce cours, l'utilisation de JavaScript sera limitée aux applications dans des navigateurs web utilisés par des humains.

Exemples d'utilisation :

- Effets visuels animés.
- Vérification des données entrées par un utilisateur.
- Communication automatique avec des appareils IoT.
- ...

Pour en savoir plus

- JavaScript sur MDN
- Une réintroduction à JavaScript sur MDN

VERSIONS

Curieusement, il est difficile de déterminer la version JavaScript utilisée par les navigateurs web. Chose incroyable, le langage n'offre pas de moyen d'obtenir cette information par programmation. On trouve des *hacks* sur internet, mais aucune solution officielle.

En utilisant un de ces *hacks*, j'ai trouvé les résultats suivants (6 janvier 2018) :

JS	Navigateur	OS
1.3	Internet Explorer 11.125.16299.0	Win10
1.5	Microsoft Edge 41.16299.15.0	Win10
1.5	Firefox 57.0.4 (64 bits)	Win10
1.5	Firefox 57.0.4 (64 bits)	macOS
1.7	Chrome 63.0.3239.132 (64 bits)	Win10
1.7	Chrome 63.0.3239.132 (64 bits)	macOS
1.7	Opera 50.0.2762.45	macOS
1.7	Opera Neon 1.0.2531.0 (64-bit)	macOS
1.7	Safari 11.0.2	macOS

On voit dans la table ci-dessus que la majorité des navigateurs modernes utilisent la version 1.7 de JavaScript. Firefox et Edge utilisent la version 1.5 et Internet Explorer est à la traîne avec la version 1.3. Ces résultats sont surprenants, parce que si on en croit la table de correspondance du site W3Schools (au bas de la page), ces versions sont très vieilles (informatiquement parlant) :

JS	ECMA	Année
1.3	1	1998
1.5	3	2000
1.7	3	2006

POSITION DES SCRIPTS

À l'instar des feuilles de style CSS, le code JavaScript peut être défini dans 3 endroits différents :

1. Dans un ou plusieurs fichiers externes.
2. Dans une ou plusieurs balises `<script>` qui peuvent se trouver dans la balise `<head>` ou dans la balise `<body>`.
3. Dans certains attributs de certains éléments HTML, comme `<body onload>` ou `<img onerror>`.

JS externe au fichier HTML

Les avantages de placer le code JS dans un ou plusieurs fichiers externes sont les mêmes que pour les fichiers CSS, c'est-à-dire que les informations ne seront téléchargées qu'une fois et mises en cache par les agents utilisateurs. De plus, plusieurs fichiers peuvent être regroupés en un seul pour minimiser le nombre de requêtes HTTP.

Voici un exemple de fichier JS :

```
"use strict";
function main(source)
{
    var now = new Date();
    console.log("Début du script " + now.getTime() +
                "\nsource = " + source);
}
main("index.js");
```

Pour faire appel à ce script, il faut inclure la balise `<script>` ci-dessous dans le fichier HTML :

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

Le plus souvent, on l'inclura juste avant la fermeture de la balise `</body>` car ceci a l'avantage de nous assurer que l'agent utilisateur a connaissance de toute la structure du fichier avant d'exécuter le script. Cependant, il est possible d'intégrer la balise `<script>` presque n'importe où dans le fichier et on la trouve souvent dans la balise `<head>`.

JS interne au fichier HTML, dans la balise `<script>`

Les avantages de placer le code JS dans un fichier qui contient les instructions JS et HTML sont les mêmes que pour le CSS (s'y référer pour plus de détails).

Voici un exemple de balises `<script>` intégrées au fichier HTML. On peut observer que ces balises peuvent être placées indifféremment dans la section `head` ou `body` et que la variable `now` et la fonction `main` définies dans le premier script sont globales et peuvent donc être réutilisées dans les scripts suivants.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset=utf-8 />
        <title>JS dans balise script</title>
        <script>
            "use strict";
            var now = new Date();
            function main(source)
            {
                console.log("fonction main " + now.getTime() +
                "\nsource = " + source);
            }
            main("head script");
        </script>
    </head>
    <body>
        <script>
            "use strict";
            main("body script");
        </script>
```

```
    </body>
</html>
```

De la même manière si une fonction ou une variable sont définies dans un fichier externe, on peut y accéder dans un script de la page HTML.

> *On remarque également que c'est la même balise `<script>` qui est utilisée que le code JS soit externe ou interne à la page HTML. Par contraste, CSS utilise la balise `<style>` pour le code interne et la balise `<link>` pour le code externe.*

JS interne au fichier HTML, dans les attributs d'un élément HTML

Certains éléments HTML possèdent des attributs qui acceptent du code JavaScript. Par exemple, dans le code ci-dessous, on voit que les éléments `<html>` et `<body>` ont respectivement leurs attributs `onclick` et `onload` qui contiennent du code JavaScript.

- `html onclick` ⇒ est exécuté lorsque l'élément `<html>` est cliqué.
- `body onload` ⇒ est exécuté lorsque l'élément `<body>` est chargé.

On voit aussi qu'un script externe est appelé (`<script src="index.js"></script>`) et que la fonction `main()` qui y est définie est utilisée par l'attribut `onload`.

De la même manière, la variable `cpt` définie dans le script à la fin de l'élément `<head>` peut être utilisée par le JS de l'attribut `onclick` car elle est globale. À priori, c'est une mauvaise idée de déclarer une variable globale après qu'elle soit utilisée comme dans cet exemple, mais l'évènement `onclick` ne sera disponible qu'une fois la page complètement chargée, donc dans ce cas, ça n'a pas d'incidence.

> *À noter que le préfixe `javascript:` est optionnel, sauf lorsqu'on utilise du JavaScript dans un attribut `href`.*

```
<!DOCTYPE html>
<html onclick="javascript:
               'use strict';
               cpt += 1;
               console.log('html onclick ' + cpt);">
<head>
    <meta charset="utf-8" />
    <title>JS dans les attributs HTML</title>
    <script src="index.js"></script>
    <script>
        "use strict";
        var cpt = 0;
    </script>
</head>
<body onload="javascript:
              'use strict';
              main('body onload');">
</body>
</html>
```

LES BASES DU LANGAGE

Casse

Le langage JavaScript est sensible à la casse, ce qui veut dire que les majuscules et les minuscules ne sont pas équivalentes.

Par exemple :

```
element = document.getElementById(id); // correct
ELEMENT = DOCUMENT.GETELEMENTBYID(ID); // incorrect
```

Fin d'instructions

Les instructions sont terminées par le caractère `;`. Les navigateurs sont souvent assez souples et tolèrent son ommission. Attendez-vous à de méchants bugs cependant, donc mieux vaut indiquer systématiquement le caractère `;`.

La directive `use strict`

```
"use strict";
```

La directive `"use strict";` se place au début d'un script et indique qu'il doit être exécuté en mode "strict" qui permet au navigateur d'exécuter le script plus rapidement. Avec `"use strict";`, la déclaration des variables est obligatoire comme nous allons le voir ci-dessous.

Pour les détails voir : le mode scrict (MDN, fr).

Déclaration des variables

Lorsqu'on utilise le mode strict, les variables doivent être déclarées et ceci ce fait généralement avec le mot clé `var`.

```
console.log( a ); // Uncaught ReferenceError: a is not defined
var a;            // Déclaration de la variable a
console.log( a ); // a = Undefined
a = 3.14;
console.log( a ); // a = 3.14
```

On peut aussi déclarer les variables avec `let` et `const`.
La portée de `let` est plus réduite que celle de `var`.
`const` n'est pas supporté par tous les agents utilisateurs, pour l'instant je vous conseille d'éviter de l'utiliser.
La plupart du temps, c'est le mot clé `var` que l'on utilisera.

Instruction let (MDN, fr)

Portée d'une variable

```
var x = 1;        // x est global
console.log( "1. x = " + x ); // x est disponible à l'extérieur de la fonction

function portée() {
    console.log( "3. x = " + x ); // x est disponible à l'intérieur de la fonction
    x = 1111;
    var y = 10;
    console.log( "4. y = " + y );
}

console.log( "2. x = " + x ); // x est disponible à l'intérieur de la fonction
portée();
console.log( "5. x = " + x ); // x est disponible à l'intérieur de la fonction
console.log( "6. y = " + y ); // x est disponible à l'intérieur de la fonction

// Les variables définies dans la fonction ne sont plus accessibles à l'extérieur.
```

Modification d'une variable avec des opérateurs arithmétiques

```
var i = 0;
i += 2;
console.log( i ); // 2
i -= 20;
console.log( i ); // -18
i *= 2;
console.log( i ); // -36
i /= 5;
console.log( i ); // -7.2
i %= 5;
console.log( i ); // -2.2
i++
console.log( i ); // -1.2
++i
console.log( i ); // -0.2
i--
console.log( i ); // -1.2
--i
console.log( i ); // -2.2
```

Pour la division entière, voir http://stackoverflow.com/a/17218003/3057377

Test `if-then-else`

```
/*
La structure "if-then-else" emprunte la syntaxe du C.
Si une condition ne comporte qu'une expression, les accolades sont optionnelles.
*/

if( true )
    console.log( "C'est vrai" );
```

```
else
    console.log( "C'est faux" );

/*
Si une condition comporte plusieurs expressions, les accolades sont obligatoires.
*/
if( false )
    console.log( "C'est vrai" );
else
{
    console.log( "C'est..." );
    console.log( "..faux" );
}
```

Comparaison faible vs stricte

```
/*
Comparaison faible vs stricte
== ⇒ Comparaison "faible" ⇒ pas de vérification du type
=== ⇒ Comparaison "stricte" ⇒ avec vérification du type

  Les_diff%C3%A9rents_tests_d_%C3%A9galit%C3%A9
*/

var a = 6;
var b = '6';

// la variable "a" contient le nombre 6 sous forme d'un nombre.

  (string).
// avec l'opérateur ==, les deux variables sont égales.
if( a == b )
    console.log( "== OK" );
else
    console.log( "== NOT OK" );

// ===
// avec l'opérateur ===, les deux variables ne sont pas égales car
// elles n'ont pas le même type.
if( a === b )
    console.log( "=== OK" );
else
    console.log( "=== NOT OK" );
```

Boucle `while`

```
// La structure "while" emprunte la syntaxe du C.
// Si elle ne comporte qu'une expression, les accolades sont optionnelles.
var compteur = 0;
while( compteur < 10 )
    console.log( "compteur = ", compteur++ );

// Attention, la variable "compteur" est définie en dehors de la boucle,
// donc la boucle suivante ne sera jamais exécutée, car "compteur == 10"
// à cet endroit du code.
while( compteur < 10 )
```

```
    console.log( "compteur = ", compteur++ );

// Si on veut isoler la variable compteur, on peut créer un bloc
{
    let compteur = 0;
    while( compteur < 5 )
        console.log( "compteur = ", compteur++ );
};

// Ici “compteur” vaut 10. Les modifications apportées à la variables déclarée
// avec “let” dans le bloc ne sont pas visbles à l’extérieur du bloc.
console.log( "compteur (après bloc) = ", compteur );
```

Quitter une boucle avec `break`

```
// On peut quitter une boucle “while” avec un “break”
compteur = 0;
while( compteur < 10 )
{
    console.log( "compteur = ", compteur++ );
    if( compteur > 5 ) break;
}
```

La boucle `do...while`

```
compteur = 0;
do
    console.log( "compteur = ", compteur++ );
while( compteur < 10 )
```

Incrémentation

```
// POST INCRÉMENTATION “X++”
// Si l’opérateur est utilisé en suffixe (par exemple : x++),
// il renvoie la valeur avant l’incrémentation.
compteur = 0;
while( compteur < 10 )
    console.log( "compteur = ", compteur++ );

// PRÉ INCRÉMENTATION “++X”
// Si l’opérateur est utilisé en préfixe (par exemple : ++x),
// il renvoie la valeur après l’incrémentation.
compteur = 0;
while( compteur < 10 )
    console.log( "compteur = ", ++compteur );

// Ça marche aussi pour la décrémentation “--”
// plus de détails ici :

  Op%C3%A9rateurs_arithm%C3%A9tiques
```

La boucle `for`

```
// BOUCLE “FOR” AVEC VAR
for( var cpt=1; cpt<=5; cpt++ ) // “++cpt” ou “cpt++” sont équivalents ici.
    console.log( "cpt = ", cpt );

// ATTENTION LA VARIABLE “CPT” EST AUSSI DÉFINIE EN DEHORS DE LA BOUCLE “FOR”
console.log( "cpt = ", cpt );

// Si on ne veut pas que la varible de compteur soit définie en dehors
// de la boucle “for”, il faut la déclarer avec “let”.
console.log( "\n\n\n# BOUCLE “FOR” AVEC LET" );
for( let cptLet=1; cptLet<=5; cptLet++ )
    console.log( "cptLet = ", cptLet );
// Ici, “cptLet” n’est plus défini.

console.log( "\n\n\n# BOUCLE “FOR” INFINIE" );
for( ;; )
{
    break;
}
```

Les fonctions

```
console.log( "\n\n\n# APPEL DE FONCTIONS" );

// On peut appeler une fonction avant qu’elle ne soit définie dans le fichier.
direBonjour();
function direBonjour()
{
    console.log( "Bonjour 1 !" );
}
direBonjour();
```

Valeur de sortie des fonctions et portée des variables

```
// Une fonction ne peut retourner qu’une seule valeur
// les variables définies dans les fonctions ne sont pas visibles à l’extérieur
function direBonjour2()
{

  pas à l’extérieur.
    return message;
}
var resultat = direBonjour2();
console.log( resultat );
console.log( typeof( message ) ); // typeof( message ) = undefined
```

Paramètres des fonctions

```
function direBonjour( prenom1, prenom2 )
{
```

```
    var message = "Bonjour, " + prenom1 + " et " + prenom2 + " !";
    return message;
}
console.log( direBonjour( "Baptiste" ) );
console.log( direBonjour( "Baptiste", "Sophie" ) );
console.log( direBonjour( "Baptiste", "Sophie", "Toto" ) );
```

Les chaînes de caractères (*string*)

```
console.log( "\n\n\n# LES CHAÎNES DE CARACTÈRES (STRING)" );

/*
En Javascript il n'y a pas de différence entre
les guillemets simples et les guillemets doubles
*/
console.log( 'guillemets simples ' );
console.log( "guillemets doubles " );
console.log( '"guillemets" \'simples\' ' );
console.log( "'guillemets' \"doubles\" " );
```

> *Attention en JSON, seuls les guillemets doubles sont valables !*

> *Il existe une nouvelle fonctionnalité appelée "Littéraux de gabarits" qui utilise le caractère "`" (backticks ou accent grave) comme délimiteur. Cette fonctionnalité est récente et n'est pas acceptée universellement.*

Concaténation de chaînes de caractères

```
console.log( "Bonjour " + "à vous !" );
console.log( "Le chiffre vaut " + 7 );
```

LES LIBRAIRIES

- https://www.jqwidgets.com/
- http://dashing.io/