



Interactions Homme-Machine

Objectif : Les connaissances de base, censées être acquises sur le WEB (complétées et unifiées à la rentrée 2020 par la mise en place du nouveau programme de SNT en classe de seconde en 2019), sont reprises ici et développées selon deux axes :

1. Les modalités des interactions Homme/Machine, qui tiennent principalement à l'effet conjoint des composants graphiques mis en place par le code HTML ; des dispositifs de style prévus par le code CSS ; du dynamisme apporté par le code Javascript.
2. L'interaction Client/serveur, que ce soit par l'intermédiaire d'une requête HTTP ou l'envoi d'un formulaire, pose toujours des questions relevant de la sécurité.

Plan du cours :

Nous nous contenterons d'une approche des trois langages utilisés dans une page WEB chez le client :

- ♦ HTML pour le texte brut et les principales balises qui permettent de le structurer
- ♦ CSS pour la mise en forme du texte selon les balises qu'il contient et les actions de l'utilisateur
- ♦ Javascript pour la réaction aux événements déclenchés par le comportement de l'utilisateur.

L'interaction avec un serveur distant est traitée principalement côté client, réservant les mécanismes du serveur (code PHP, requête SQL vers une base de données) à la terminale.

Introduction historique :

Le WEB (de l'anglais *web*, toile d'araignée) désigne un système donnant accès à un ensemble de données reliées par des liens hypertextes et accessibles sur le réseau internet. Les textes, photos, vidéos, graphiques, sons, programmes qui circulent sur le WEB sont exprimés dans des formats normalisés par le W3C (*World Wide Web Consortium*).

1965 : Invention du concept d'hypertexte par Theodor Holm Nelson, dit Ted Nelson (1937-).

1989 : Création du World Wide Web par Timothy John Berners-Lee, dit Tim Berners Lee (1955-) durant ses travaux au CERN (Conseil Européen pour la Recherche Nucléaire) à Genève. TBL crée le premier navigateur/éditeur WEB et le premier serveur ; de plus, il est considéré comme l'inventeur¹ du langage HTML. Il dirige depuis 1994 le W3C qu'il a créé.

1993 : Le CERN verse dans le domaine public ses logiciels WEB. Le navigateur Mosaic, développé par Eric Bina et Marc Andreessen au NCSA (National Center for Supercomputing Applications), dans l'Illinois, jette les bases de l'interface graphique des navigateurs modernes en intégrant les images au texte ce qui entraîne un accroissement exponentiel de la popularité du WEB. Certains développeurs de Mosaic, créeront ensuite le navigateur Netscape qui, d'évolutions en évolutions, est devenu aujourd'hui Mozilla Firefox.

1994 : Apparition progressives des technologies pour le développement de site Web interactif : Håkon Wium Lie (1965-) élabore, avec Bert Bos, le concept de feuilles de style en cascade (CSS). Rasmus Lerdorf crée le langage PHP pour son site WEB.

1995 : Invention du langage JavaScript par Brendan Eich (1961-), parfois considéré comme l'une des technologies cœur du WEB.

1. Il fut aidé à ses débuts par l'ingénieur et informaticien Robert Cailliau qui cosigna notamment avec lui, en novembre 1990, un document désormais entré dans l'Histoire et intitulé « WorldWideWeb : Proposition pour un projet hypertexte »

1. Structuration HTML

Les suites de caractères, encodés en ASCII, *latin – 1* ou *utf – 8*, sont des textes bruts. Les logiciels de traitement de texte les enrichissent pour introduire des qualifications particulières (soulignage, italique, etc.), les structurer (en les subdivisant en chapitres, parties, paragraphes, etc. ou en formant des tables ou des listes), apporter des informations sur le texte (auteur, date, etc.), créer des liens entre certaines parties du texte ou avec d'autres textes, ajouter des images, etc. Les formats spécifiques de ces textes enrichis peuvent être fermés (le propriétaire en garde le secret, comme doc de Microsoft) ou ouverts².

Ainsi, les pages WEB sont écrites dans le langage de balises HTML (*HyperText Markup Language*).

Les balises vont deux par deux, comme les parenthèse : après une balise ouvrante qui s'écrit `<nameBalise>`, il doit toujours y avoir balise fermante qui s'écrit `</nameBalise>`.

On peut emboîter des balises les unes dans les autres mais pas les faire se chevaucher.

Si on veut mettre un passage en italique par exemple, on va utiliser les balises `<i>` et `</i>` ; si on veut mettre un passage en gras, il faut utiliser les balises `` et `` (b pour *bold*).

Le texte HTML (*HyperText Markup Language*) s'écrit donc en HTML :

```
HTML (<i><b>H</b>yper<b>T</b>ext <b>M</b>arkup <b>L</b>anguage</i>).
```

Depuis 2007, la version du HTML qu'il faut utiliser est le HTML5. Une page HTML5 se présente toujours avec le squelette ci-dessous. Les indentations ne sont pas obligatoires ; elles n'ont pour seul but que d'améliorer la lecture.

À ce sujet, je signale que mes présentations de codes sont volontairement compactes et ne respectent pas toujours les conseils de bonne présentation. J'espère que ce gain de concision compense la baisse éventuelle de lisibilité.

```
<!DOCTYPE html>
<html>
  <head>
    <title> titre </title>
  </head>
  <body>
    <!-- Contenu de la page -->
  </body>
</html>
```

La balise `<!DOCTYPE html>` précise que le texte qui suit est du HTML5.

Les balises `<html>` et `</html>` délimitent la page HTML. Celle-ci contient :

- ♦ Les balises `<head>` et `</head>` délimitent l'en-tête qui contient les informations sur le document, notamment son titre (écrit entre les balises `<title>` et `</title>`) et éventuellement des métadonnées. On peut par exemple ajouter la balise `<meta charset='utf8' />` qui est autofermante car il n'y a pas de contenu à disposer dans une paire ouvrante/fermante. Cette balise contient une métadonnée précisant le système d'encodage.
- ♦ Les balises `<body>` et `</body>` délimitent le corps ou contenu de la page qui contient ce qui est affiché. La ligne `<!-- Contenu de la page -->` est un commentaire. Tout ce qui est écrit entre `<!--` et `-->` est du commentaire : il ne sera pas lu par le navigateur, chargé de représenter ce qui est spécifié par la page HTML.

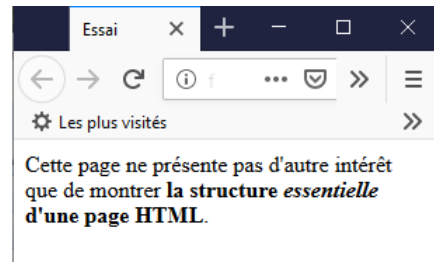
Voici un exemple simpliste montrant le texte d'un fichier « `htmlEssai.html` » à gauche et sa réalisation dans un navigateur à droite. On remarquera que le texte entré entre les balises `<title>` et `</title>` est celui qui s'affiche sur l'onglet de la fenêtre. C'est également le nom que stockera le gestionnaire de marque-pages du navigateur si on lui demande de mémoriser l'adresse (URL) de cette page.

2. Un format ouvert est gratuit, sa documentation est disponible en ligne et suffisante pour développer une implémentation complète. Les formats `.txt` (texte en ASCII), `.tex` (texte en \LaTeX , le format de ce document), `.odt` (texte pour le logiciel OpenOffice), `.html` (texte pour le WEB), `.css` (feuilles de style CSS) `.js` (code javascript) sont ouverts

```

<!DOCTYPE html>
<html>
  <head>
    <title> Essai </title>
  </head>
  <body>
    Cette page ne présente pas d'autre intérêt que de montrer
    <b>la structure <i>essentielle</i> d'une page HTML</b>.
  </body>
</html>

```



La syntaxe générale d'une balise est `<name attribut1='valeur' attribut2='valeur'...>` où `name` est le nom de la balise et `attribut1`, `attribut2` sont des attributs, des options, dont il faut choisir les valeurs. La balise `<html>` admet un seul attribut : `lang`. Pour indiquer que la langue utilisée est `fr`, soit du français, on écrit `<html lang="fr">`. Cela peut servir dans certains cas, pour lire la page avec un logiciel de synthèse vocale. Dans d'autres cas, la liste des attributs possibles est plus importante³. La balise d'entête `meta` par exemple en accepte 19 dont 4 seulement sont spécifiques à cette balise (`name`, `http-equiv`, `content` et `charset`), les autres pouvant se retrouver dans d'autres balises (`accesskey`, `class`, `contenteditable`, `contextmenu`, `dir`, `draggable`, `dropzone`, `hidden`, `id`, `lang`, `spellcheck`, `style`, `tabindex`, `title` et `translate`).

Il est assez instructif de consulter le code HTML d'une page WEB existante. Il y a généralement un outil de votre navigateur qui le permet : sur Mozilla Firefox, il faut ouvrir le « menu » en haut à droite (l'icône consiste en 3 traits horizontaux superposés), choisir « Développement web », puis « Code source de la page ». Voici par exemple l'entête d'une page concernant le sujet⁴ Introduction au WEB en NSI par David Roche dont nous traitons.

```

1 <!doctype html>
2 <!-- Auteur : David Roche @davR74130 -->
3 <html lang="fr">
4   <head>
5     <meta charset="utf-8">
6     <title>Introduction au Web</title>
7     <link rel="stylesheet" href="css/style.css">
8     <script src="highlight/highlight.pack.js"></script>
9     <script>hljs.initHighlightingOnLoad();</script>
10    <script src="./css/js/vendor/jquery.min.js"></script>
11    <script src="./css/js/flat-ui.min.js"></script>
12  </head>

```

a. Catalogue de balises

Balises d'entête

`<title> ... </title>` : Titre de la page

`<style> ... </style>` : Permet d'inclure du code CSS

`<link />` : Permet d'indiquer certaines informations comme :

- ✦ Inclure une page de style `<link rel="stylesheet" href="style.css"/>`
- ✦ Déclarer un icône pour le site :
`<link rel="shortcut icon" type="image/x-icon" href="icon.ico"/>`

`<script> ... </script>` : Permet d'inclure un script comme :

- ✦ Du code : `<script type="text/javascript"> ... </script>`
- ✦ Un fichier script : `<script type="text/javascript" src="script.js"/>`

`<meta ... />` : Permet de définir différentes propriétés comme :

- ✦ L'auteur de la page : `<meta name="author" content="Philippe Moutou"/>`
- ✦ La description de la page : `<meta name="description" content="Cours NSI"/>`
- ✦ Les mots-clé de la page : `<meta name="keywords" content="Web,HTML,CSS,Javascript"/>`
- ✦ Une adresse de contact : `<meta name="reply-to" content="ph.moutou@free.fr"/>`
- ✦ La spécification du système d'encodage des caractères : `<meta charset="utf-8"/>`
- ✦ L'instruction d'empêcher la mise en cache de la page :
`<meta http-equiv="pragma" content="no-cache"/>`

3. Pour des spécifications en français plus complètes que ce cours, reportez-vous sur les multiples propositions du WEB, par exemple le site de Jacques MARTINET <https://jaetheme.com/balises-html5/#html>

4. Introduction au WEB en NSI par David Roche https://pixees.fr/informatiquelycee/n_site/

Balises de contenus

`<p> ... </p>` : Paragraphe

`<pre> ... </pre>` : Paragraphe affiché tel qu'il a été tapé dans le code (avec les espaces notamment)

`<h#> ... </h#>` : Titre de niveau # : `<h1>` (le plus important), ..., `<h6>` (le moins important)

`<a> ... ` : Permet de définir un hyperlien

- ✦ Pour pointer vers la page d'adresse `url` : ` ... `.

Un nom précédé par un dièse (#) indique une cible interne au document (un ID).

La cible est repérée grâce aux balises ` ... ` (voir l'exemple).

- ✦ Pour charger la réponse dans une nouvelle fenêtre utiliser l'option `target="_blank"`

`
` : Passage à la ligne

`<hr/>` : Ligne de séparation horizontale

`` : Insertion d'une image

- ✦ `` insère l'image `image1.png` située dans le dossier `images` du même dossier que la page. La valeur de `alt` est le texte alternatif utilisé lorsqu'il est impossible d'afficher l'image.
- ✦ Les attributs `width` et `height` donnent les dimensions de l'image en pixels, ce qui permet de redimensionner. Par exemple `` redimensionne l'image à `150px` de haut sur `200px` de large.

`<div> ... </div>` : Création d'un bloc de contenu de type *block* (les `div` se positionnent les uns *au-dessus* des autres comme `h1`, `h2`, `p`, `pre`, `ul`, `ol`, `li`, `table`, etc.).

` ... ` : Création d'un bloc de contenu de type *in-line* (les `span` se positionnent les uns *à côté* des autres comme `a`, `img`, `em`, `input`, `label`, etc.).

`^{...}` : Mise en exposant

`_{...}` : Mise en indice

` ... ` : Passage sur lequel porte une emphase accentuée.

Exemple de liens interne et externe : l'appel à la note du bas est ici doublé par un appel retour du bas vers le haut (mécanisme permettant de faciliter la lecture d'une note de bas de page, sans perdre le fil du texte).

```
<p>Dans un long<a href="#note" name="retour"><sup>1</sup></a> texte ,
il y a parfois des indications à apporter comme une référence ou une anecdote.
Cette indication créant une sorte de rupture dans le texte peut être reportée
en bas de page, ce que nous avons fait ici.<br/> De nombreux exemples peuvent
se voir sur <a href="http://ph.moutou.free.fr/">cette page</a>, certains liens
pointant vers d'autres pages et d'autres pointant vers des documents.</p>
<a href="#retour" name="note"><sup>1</sup></a>Tout est relatif,
ici le texte n'est vraiment pas long.
```

```
Dans un long1 texte , il y a parfois des indications à apporter
comme une référence ou une anecdote. Cette indication créant une
sorte de rupture dans le texte peut être reportée en bas de page, ce
que nous avons fait ici.
De nombreux exemples peuvent se voir sur <a href="#">cette page, certains liens
pointant vers d'autres pages et d'autres pointant vers des documents.
```

¹Tout est relatif, ici le texte n'est vraiment pas long.

Listes et Tableaux

` ... ` : Liste à puces non numérotée

` ... ` : Liste à puces numérotée (option par défaut `:type="1"`)

- ✦ Ajouter l'option `type="A"` pour un numérotation alphabétique majuscule
- ✦ Ajouter l'option `type="I"` pour un numérotation romaine majuscule

` ... ` : Item de liste à puces (numérotée ou non)

`<dl> ... </dl>` : Liste de définitions

`<dt> ... </dt>` : Titre de la définition

`<dd> ... </dd>` : Définition

`<table> ... </table>` : Déclaration d'un tableau

`<caption> ... </caption>` : Titre du tableau

`<tr> ... </tr>` : Ligne du tableau

`<td> ... </td>` : Case du tableau (`<th> ... </th>` pour une case d'entête, par défaut en gras)

- ✦ `<td colspan=n>` : la case occupe `n` colonnes
- ✦ `<td rowspan=n>` : la case occupe `n` lignes

Voici, par exemple, un extrait de page concernant les listes et tableaux. Une mise en forme plus poussée du tableau relèverait d'éléments de style que volontairement nous n'avons pas implémentés ici (pour les réserver aux feuilles de style CSS, voir plus loin), exceptée l'option `border=5` qui ajoute une bordure.

```
<h1>Trois types de listes</h1>
<h2>Liste non numérotée</h2>
<ul>
  <li>Premier élément de ma liste non numérotée</li>
  <li>Second élément de ma liste non numérotée</li>
</ul>
<h2>Liste numérotée</h2>
<ol type="I">
  <li>Premier élément de ma liste numérotée</li>
  <li>Second élément de ma liste numérotée</li>
</ol>
<h2>Liste de définitions</h2>
<dl>
  <dt>Définition 1</dt>
  <dd> Texte de la définition 1</dd>
  <dt>Définition 2</dt>
  <dd> Texte de la définition 2</dd>
</dl>
<h1>Un exemple de tableau</h1>
<table border=5>
  <caption>Petit tableau simple</caption>
  <tr><td></td>
    <th>Donnée</th>
    <th>Commentaire</th>
  </tr>
  <tr>
    <th>n°1</th>
    <td>Donnée 1</td>
    <td>Commentaire 1</td>
  </tr>
  <tr>
    <th>n°2</th>
    <td>Donnée 2</td>
    <td>Commentaire 2</td>
  </tr>
</table>
```

Trois types de listes

Liste non numérotée

- Premier élément de ma liste non numérotée
- Second élément de ma liste non numérotée

Liste numérotée

- I. Premier élément de ma liste numérotée
- II. Second élément de ma liste numérotée

Liste de définitions

Définition 1
Texte de la définition 1

Définition 2
Texte de la définition 2

Un exemple de tableau

Petit tableau simple

	Donnée	Commentaire
n°1	Donnée 1	Commentaire 1
n°2	Donnée 2	Commentaire 2

Formulaires

`<form> ... </form>` : Déclaration d'un formulaire.

Les attributs obligatoires de cette balise sont

- ♦ `method="post"` ou `method="get"`, selon le type de transmission que l'on souhaite (avec `get`, on est limité dans la quantité de données pouvant être envoyées et surtout, celles-ci vont être envoyées en clair).
- ♦ `action=...` pour préciser l'adresse de destination du formulaire (souvent une page PHP d'un serveur qui effectuera un traitement et retournera une réponse au client).

`<input .../>` : Champ d'un formulaire (associé ou non à un `label`).

L'attribut `type` est obligatoire :

- ♦ `type="text"` : le champ est une zone de texte
- ♦ `type="password"` : le champ est une zone de texte où les caractères sont masqués
- ♦ `type="checkbox"` : le champ est une case à cocher
- ♦ `type="radio"` : le champ est un bouton radio
- ♦ `type="submit"` : le champ est un bouton d'envoi
- ♦ `type="reset"` : le champ est un bouton de remise à zéro des contenus des champs

Un autre attribut obligatoire pour les champs d'un formulaire : l'attribut `name` qui va permettre, par la suite, de reconnaître l'élément du formulaire associé à chaque donnée envoyée.

`<label> ... </label>` : Libellé d'un champ du formulaire. Le texte du libellé peut être techniquement associé à un champ (un lecteur d'écran pourra énoncer le libellé pour permettre une meilleure expérience) en utilisant les attributs `for` (dans le `label`) et `id` (dans le `input`).

`<textarea> ... <textarea/>` : Zone de saisie multiligne.

`<select> ... <select/>` : Liste déroulante dont les éléments sont déclarés par

`<option> ... <option/>`

Exemple de formulaire montrant la plupart de ces éléments, structuré par l'intermédiaire d'un tableau. Le mécanisme d'un formulaire sera envisagé ultérieurement car il relève de l'interaction client/serveur. Disons seulement qu'en pressant le bouton de type `submit`, le formulaire sera envoyé à l'adresse entrée dans l'attribut `action` de la balise `<form>`.

```
<table>
<form name="formulaire" method="post" action="https://www.monSite/traitement.php">
<tr><td><label for="Nom">Nom :</label></td>
  <td><input type="text" name="nom" id="Nom" size="30" maxlength="50"></td></tr>
<tr><td><label for="Mdp">Mot de passe :</label></td>
  <td><input type="password" name="mdp" id="Mdp" size="30" maxlength="50"></td></tr>
<tr><td>Sexe :</td><td>
  Homme : <input type="radio" name="sexe" checked value="M"></br>
  Femme : <input type="radio" name="sexe" value="F"></td></tr>
<tr><td>Fonction :</td><td>
  <select name="fonction">
  <option value="enseignant">enseignant</option>
  <option value="lyceen">lycéen</option></select></td></tr>
<tr><td>Commentaires :</td><td>
  <textarea rows="3" name="commentaires">Vos commentaires</textarea></td></tr>
<tr><td colspan="2">
  <input type="submit" value="Envoyer">
  <input type="reset" value="Rétablir"></td></tr>
</table>
<input type="checkbox" name="rester" value="true">Rester connecté
</form>
```

2. Mise en page CSS

CSS est le diminutif de *Cascading StyleSheets*, ou feuilles de style en cascade. Ce langage, créé en 1996, a pour rôle de mettre en forme du contenu en lui appliquant ce qu'on appelle des styles. Il permet, par exemple, de définir la taille, la couleur ou l'alignement d'un texte. On dispose généralement tout le code CSS relatif à un fichier HTML dans un fichier d'extension `css`. Cette page CSS peut ainsi être partagée entre les différentes pages d'un site internet, afin d'en assurer une homogénéité de style.

Pour qu'un code HTML soit mis en forme par du code CSS extérieur, il faut lier les deux codes avec une déclaration `<link rel="stylesheet" href="style.css"/>` dans l'entête de la page HTML. Il est possible, dans des cas simples, de se passer d'un fichier CSS en mettant tous les dispositifs de style entre des balises `<style>...</style>` de la page HTML. Il est même possible de mettre du code CSS à l'intérieur d'une balise quelconque de la page HTML. En réalité, toutes ces trois possibilités sont exploitables simultanément, le navigateur sait s'y retrouver pour déterminer le style à appliquer à un élément. À cet effet, il utilise les règles de priorités du CSS.

Pour apporter le style à une page WEB, le navigateur peut avoir des indications qui proviennent de différentes sources : du navigateur lui-même (réglées par les préférences de l'utilisateur), de l'auteur de la page WEB ou de l'utilisateur (transmises au site WEB par l'intermédiaire de formulaires). Les règles de priorités appliquées en cas de conflit⁵ sont, par ordre croissant :

1. la feuille de style par défaut du navigateur (styles *agent utilisateur*)
2. les styles de la page (styles *auteur*), qu'ils soient internes ou externes ou *inline*
3. les styles définis par l'utilisateur (styles *utilisateur*)

À l'intérieur de ces trois groupes, les styles sont pondérés par la priorité des sélecteurs. Les styles *auteur inline* ont la plus forte priorité possible. Si deux sélecteurs s'appliquant à un élément ont la même priorité, c'est celui qui vient en dernier qui a le dessus. Toutes ces règles expliquent l'utilisation du mot « cascade » attribué aux feuilles de style. Les sélecteurs de style pour un élément auront beau être nombreux, l'ensemble des règles conduit à n'en choisir en définitive qu'un seul. Comme notre but n'est pas de devenir expert en conception de sites WEB, nous en resteront là pour ces questions de priorités.

Voici un premier exemple de code CSS qui indique que les titres `<h1>` doivent avoir une couleur rouge sur fond jaune et une taille de 14px tandis que les paragraphes `<p>` doivent être bleus sur fond vert et soulignés. Ces choix ne semblent pas très esthétiques ; ils ne servent qu'à introduire la feuille de style.

5. Se reporter à la spécification https://openweb.eu.org/articles/cascade_css/

```

<html>
  <head>
    <title> Essai </title>
    <meta charset="utf-8"/>
    <link rel="stylesheet" href="testStyle.css"/>
  </head>
  <body>
    <div>
      <h1>Les grandes idées du CSS</h1>
      <p>L'idée essentielle du CSS est l'apport d'un style au texte HTML,
      en dehors de toute préoccupation de contenu.</p>
      <p>Chaque élément peut faire l'objet d'injonctions de styles issues
      de divers endroits (navigateur, auteur, utilisateur).</p>
    </div>
  </body>
</html>

```

```

testStyle.css
div {border       : 5px solid black;
     padding      : 10px;
     color        : white;
     background-color : gray;
}
h1 {font-size    : 1.6em;
     color        : red;
     background-color : black;
}
p {font-size     : 0.8em;
   color         : blue;
   background-color : white;
}

```



La syntaxe générale des commandes de style sont : `selecteur{propriété: valeur1, valeur2, ...;}` (majuscules ou minuscules, peu importe : le code CSS est insensible à la casse).

Par exemple `h1{font-size:18px;color:blue;}` (ne pas oublier les point-virgules !)

Le sélecteur est, au choix :

- ♦ un des éléments du code HTML : `p`, `h1`, `div`, etc.
 - On peut en mettre plusieurs séparés par une virgule, par exemple `h1,h2`.
- ♦ `*` : le sélecteur universel (les styles s'appliquent alors à tous les éléments)
- ♦ `#x` où `x` est un identifiant de l'élément (spécifié par l'attribut HTML `id="x"`).
 - Par composition, cela peut donner `p#x` : tout élément `p` d'identifiant `x`.
- ♦ `.y` où `y` est une classe (spécifiée par l'attribut HTML `class="y"`).
 - Par composition, cela peut donner `p.y` : tout élément `p` de classe `y`.

Il y a des raffinements pour le choix du sélecteur, je vous renvoie à la documentation⁶.

- ♦ On peut préciser que seuls les attributs `attr` d'un élément `e` seront affectés en utilisant le sélecteur `e attr` (avec un espace sans virgule).
- ♦ Concernant un élément graphique `e`, on peut préciser : `e:link` (élément n'ayant pas encore été visité), `e:visited` (élément ayant déjà été visité), `e:active` (élément activé), `e:hover` (élément pointé), `e:focus` (élément recevant l'attention).

On peut ajouter des commentaires dans le code CSS avec la syntaxe `/* ... */`.

Cela permet, en phase d'essai, de supprimer l'effet d'un ensemble de commandes.

Par exemple `/*div{font-weight:bold;font-style:italic;}*/` supprime l'action des deux règles de style sur les blocs `div` ou `div{font-weight:bold; /*font-style:italic; */}` supprime seulement l'action de la règle portant sur le style des polices `font-style` du bloc.

a. Catalogue de règles CSS

Propriétés des textes

font-family : Permet de spécifier la police à utiliser en priorité, puis une autre et une dernière au cas où les deux premières feraient défauts. Exemple : Arial, Verdana, sans-serif.

font-size : Permet de spécifier la taille de la police à utiliser. Cette taille s'exprime dans différentes unités : `12px` (12 pixels), `1.2em` (1,2 fois la taille définie dans l'élément parent), `80%` (80% de la taille normale) ; on peut aussi l'indiquer avec un mot-clé : `small`, `medium` ou `large` (tailles absolues) ou `smaller`, `larger` (tailles relatives).

font-align : Permet de justifier le texte, les attributs étant `justify`, `left`, `right` ou `center`.

color : La couleur de la police peut être donnée en code RGB décimal `rgb(214, 122, 127)` ou hexadécimal précédé d'un dièse `#00ff00` (vert), ou bien avec un nom répertorié comme `red` ou `rebeccapurple` (voir la liste des couleurs⁷).

text-decoration : Pour souligner un texte, utiliser `underline` (`underline dotted` pour le souligner en pointillés, `green wavy underline` pour le souligner avec des vagues vertes,

6. Par exemple <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

7. couleurs HTML : https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#Color_keywords

`underline overline #FF3028` pour le surligner et le souligner en rouge), pour le barrer `line-through`.

`font-weight` : Permet de modifier la graisse, pour des caractères plus gras `bold` ou moins gras `light`.

`font-height` : Permet d'espacer les lignes entre elles, par exemple en pourcentage de l'espacement normal `150%`.

`font-style` : Permet de changer la forme des caractères, les attributs étant `normal`, `italic` ou oblique `10deg` (police oblique inclinée de `14deg` par défaut).

Voici une brève illustration de quelques décorations apportées par l'intermédiaire de classes.

```
<p class="under">Ce texte est souligné <span class="line"> même cette partie barrée</span>, ce qui en montre l'importance.</p>
<p class="underover">Ce texte étant vraiment important a été souligné <em>et</em> surligné.</p>
<p>Ce <a class="plain" href="#">lien</a> a été privé de son soulignement et de sa couleur habituelle pour le dissimuler</p>
```

```
.under {text-decoration: underline red;}
.underover {text-decoration: dashed underline overline;}
.line {text-decoration: line-through;}
.plain {text-decoration: none ;color:grey;}
.barre {text-decoration: line-through;}
```

<p>Ce texte est souligné même cette partie barrée, ce qui en montre l'importance.</p> <p>Ce texte étant vraiment important a été souligné et surligné.</p> <p>Ce lien a été privé de son soulignement et de sa couleur habituelle pour le dissimuler</p>
--

Propriétés des tableaux

Les règles générales s'appliquent pour tous les éléments d'un tableau quelconque.

En particulier, on peut sélectionner un ou plusieurs éléments parmi `table`, `td`, `tr`, `th`, `caption`.

De nombreux attributs de ces éléments sont des attributs généraux concernant les couleurs `background-color` (couleur de fond), `color` (couleur des caractères), concernant les bordures `border`, concernant les marges du bloc `padding` ou concernant les polices `font-family`, etc.

Certains attributs sont spécifiques aux balises de tableaux (`table`, etc.) :

- ♦ `caption-side` spécifie à quel endroit doit être placé le titre du tableau : `top` (au-dessus) ou `bottom` (en-dessous).
- ♦ `border-collapse` spécifie si les bordures du tableau doivent être séparées (`separate`, par défaut) ou fusionnée (`collapse`).
- ♦ `empty-cells` pour spécifier comment doivent apparaître les cases du tableau qui sont laissées vides : `show` (montrer) ou `hide` (cacher). Cette propriété n'a un effet que si `border-collapse` a l'attribut `separate`.
- ♦ `border-spacing` pour spécifier l'écart entre les cases si `border-collapse` a l'attribut `separate`. On peut mettre deux valeurs, la 1^{re} pour l'écart horizontal et la 2^e pour l'écart vertical (`border-spacing: 1cm 2em`) ou une seule qui règle les deux.
- ♦ `vertical-align` pour spécifier comment est disposé le texte dans la cas : au centre `middle`, en haut `top` ou en bas `bottom` (NB : cet attribut n'est pas spécifique de `table`, il est également utilisable pour tous les éléments *inline*).
- ♦ `table-layout` indique quel algorithme doit être utilisé pour définir les dimensions des lignes et colonnes : `auto` (par défaut) pour s'adapter au contenu ou `fixed` pour obtenir des espacements constants. Régler l'attribut `width`, par exemple `width:150px` ou `width:100%`.

EXEMPLE 1 – Pour illustrer ce que le style peut apporter à un tableau, réalisons un squelette de jeu « memory ». Des lettres sont dissimulées par paires dans les cases d'un tableau (rendues invisibles car la couleur du fond est la même que celle des caractères). Lorsqu'on clique sur une case, on voit apparaître la lettre sur un fond de couleur la caractérisant ; quand on ne clique plus la lettre redevient cachée.

Ce jeu, dans cet état, n'est pas finalisé il reste à ajouter du dynamisme : quand on a reconnu une paire de lettres, celles-ci disparaissent du plateau ; à la fin d'une partie, les lettres sont à nouveaux disposées, dans un ordre aléatoire. Pour apporter ce dynamisme, il faudra utiliser du code javascript (voir plus loin).


```

<table>
  <tr><td class="B">B</td><td class="A">A</td>
  <td class="C">C</td><td class="E">E</td></tr>
  <tr><td class="H">H</td><td class="G">G</td>
  <td class="A">A</td><td class="D">D</td></tr>
  <tr><td class="E">E</td><td class="G">G</td>
  <td class="B">B</td><td class="H">H</td></tr>
  <tr><td class="D">D</td><td class="F">F</td>
  <td class="C">C</td><td class="F">F</td></tr>
</table>

```

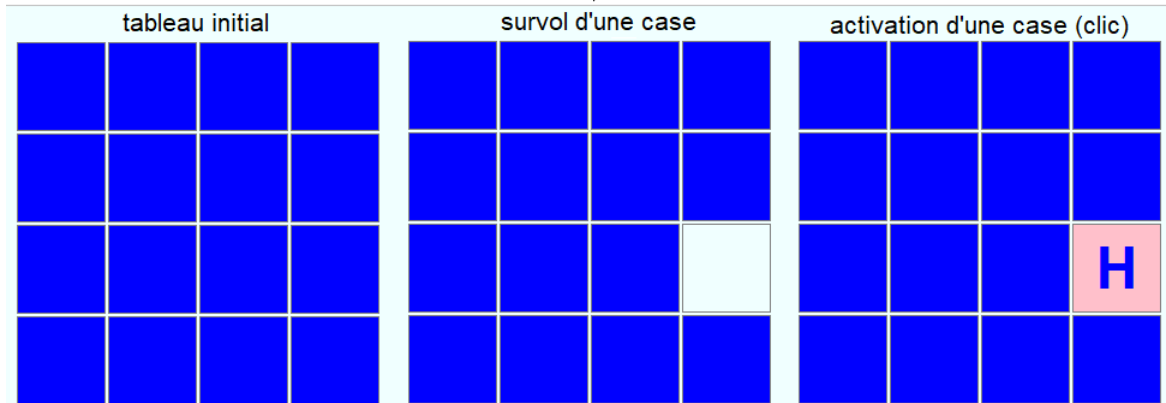
B A C E
H G A D
E G B H
D F C F

```

body{background-color: azure;
  font-family: sans-serif;font-size: 100%;}
td {background-color: blue; color:blue ;
  font-weight:bold ; font-size: 300% ;
  border: 1px solid Gray; height: 70px ;
  text-align: center ; width:70px ;}
td:hover {background-color: azure ;color:azure;}
.A:active{background-color: red ;color:blue;}
.B:active{background-color: yellow;color:blue;}
.C:active{background-color: grey ;color:blue;}
.D:active{background-color: black ;color:blue;}
.E:active{background-color: white ;color:blue;}
.F:active{background-color: green ;color:blue;}
.G:active{background-color: brown ;color:blue;}
.H:active{background-color: pink ;color:blue;}

```

le tableau sans le code css ↑
le tableau avec le code css ↓



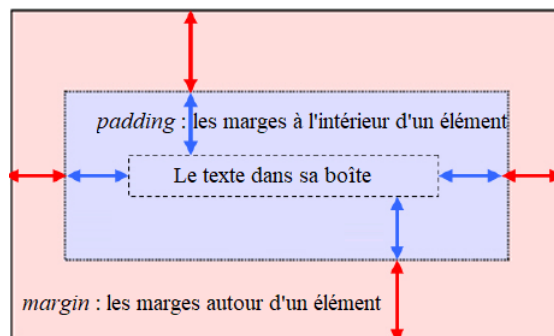
Propriétés de positionnement

Pour bien comprendre la notion de marge extérieure à un bloc (**margin**) ou intérieure à un bloc (**padding**), il faut se reporter à la figure ci-dessous qui montre un bloc. Sachant que tous les éléments HTML sont inclus dans un bloc, on réalise le nombre de combinaisons possibles pour les différentes définitions de marges. Heureusement, on peut ne rien préciser du tout concernant ces marges, le navigateur se chargeant de les définir au mieux à l'aide de ses propres critères ainsi que des valeurs par défaut des composants HTML. Autre précision : les spécifications de dimension ne fonctionnent qu'avec des éléments de type *block* (voir plus haut).

Si on veut raffiner les réglages de marges, on peut les définir chacune séparément : **margin-top**, **margin-right**, **margin-bottom** et **margin-left** (respectivement haut, droite, bas et gauche). Si on donne uniquement une valeur à **margin**, elle s'applique aux quatre marges ; si on en donne deux, la 1^{re} s'applique aux marges haut et bas, la 2^e aux marges droite et gauche. Si on donne quatre valeurs, elles s'appliquent dans l'ordre des aiguilles d'une montre : top-right-bottom-left.

Ce qu'on vient de dire pour les marges extérieures s'applique également aux marges intérieures⁸ : on règle **padding-top**, **padding-right**, **padding-bottom** et **padding-left** ensemble ou séparément.

Un bloc HTML n'est pas seulement doté de marges : il a aussi une hauteur (**height**), une largeur (**width**), une hauteur et une largeur maximum (**max-height** et **max-width**), une hauteur et une largeur minimum (**min-height** et **min-width**). Il a aussi un attribut de visibilité **visibility** qui peut être **visible**, **hidden** (le bloc n'est plus visible mais sa place est conservée) ou **collapse** (le bloc n'est plus visible et sa place disparaît).



8. Voir les différents effets sur la page <https://developer.mozilla.org/en-US/docs/Web/CSS/padding>

Les réglages de marges permettent de positionner un bloc dans le flot, relativement aux autres, en créant des espacements horizontaux et verticaux. On peut également intervenir sur la position d'un bloc dans la page : les éléments de type *block* se positionnent les uns au-dessous des autres alors que les éléments de type *inline* se positionnent les uns à côté des autres. On ne peut mettre des éléments *block* (comme une `div`) à l'intérieur d'un élément *inline* (comme une `span`). Il est possible de modifier le type d'un élément avec l'attribut `display`. Si celui-ci est mis à `none`, l'élément disparaît de l'affichage (ainsi que tout ce qu'il contient).

Le type *inline-block* existe également. Un élément de ce type se comporte extérieurement comme un élément *inline* mais intérieurement comme un *block*, exactement comme une image : on peut alors spécifier son alignement par rapport au reste du flot, en donnant une valeur à l'attribut `vertical-align` qui peut être `middle` (par défaut), `top` ou `bottom`.

Si un contenu dépasse de la boîte qui le contient (cela arrive pour des cases de tableau contenant un texte trop long, ou lorsqu'on utilise un texte préformaté `pre`), pour masquer ce qui dépasse, on peut régler l'attribut `overflow` à `auto` pour laisser le navigateur faire au mieux, ou bien :

- ♦ le rendre visible avec la valeur `visible`
- ♦ le masquer sans possibilité de lecture avec `hidden`
- ♦ le masquer avec possibilité de lecture par des ascenseurs avec `scroll`

Les positions des blocs se succèdent comme indiqué plus haut, mais un bloc peut sortir de ce flot.

Pour cela, il faut régler l'attribut `position` qui est, par défaut à `static`.

- ♦ En donnant la valeur `relative`, les coordonnées sont calculées par rapport au conteneur parent. Par exemple, `position:relative;left:50px;top:200px;`
- ♦ En donnant la valeur `absolute`, l'élément est retiré du flot et positionné relativement au coin supérieur gauche du parent positionné (non `static`) le plus proche sans tenir compte des marges de celui-ci.
- ♦ La position `fixed` est liée à la fenêtre d'affichage du navigateur : l'élément ne bouge pas quand défile le texte de la page (utile pour un logo, une signature) en restant en-dessus ou en dessous, selon la valeur de l'attribut `z-index`.

On peut découper un rectangle dans l'élément qui seul sera affiché, le reste étant à sa place mais invisible. L'attribut `clip` s'utilise dans ce but, pour les éléments positionnés de façon absolue. La syntaxe donne successivement les valeurs `y1`, `x2`, `y2`, `x1` d'un rectangle dont le sommet supérieur gauche a pour coordonnées (`x1,y1`) et le sommet inférieur droit a pour coordonnées (`x2,y2`), par exemple `clip:rect(50px, 250px,200px,20px)`. Cette caractéristique permet de donner des effets dynamique au contenu, par exemple en jouant sur l'état de l'élément (`hover`, `active`, etc.).

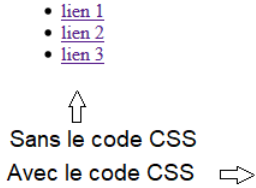

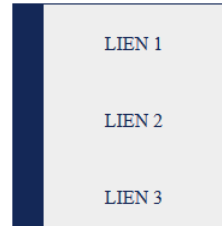
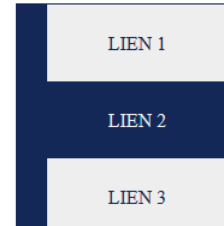

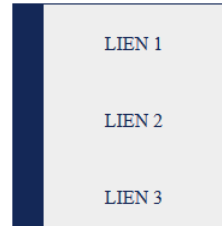
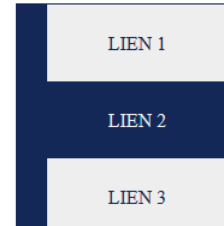

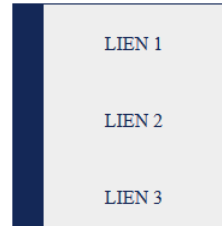
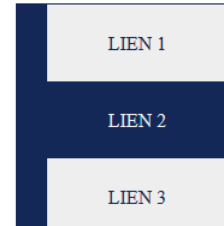
On peut créer des boîtes flottantes, les autres éléments se positionnant autour d'elles. En réglant l'attribut `float` à `left`, par exemple, la boîte va flotter à gauche. Des marges sont applicables à cet élément flottant qu'on peut préciser d'un coup dans l'ordre horaire. Voici un exemple qui s'appliquera aux images `img` d'un paragraphe `p`, leur attribut un placement flottant à gauche avec des marges et une bordure noire en trait plein :

```
p.
```

Les bordures d'un élément définissent son pourtour. Elles possèdent une épaisseur `border-width`, un style `border-style` et une couleur `border-color`. Pour les styles, on a le choix entre `dashed` (tirets), `dotted` (pointillés), `solid` (traits pleins), `inset` et `outset` (avec effet de perspective), `double` (deux traits), `groove` (en relief).


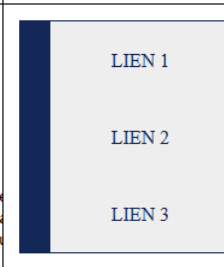
EXEMPLE 2 – Pour montrer l'effet dynamique apporté par la découpe d'un `clip`, construisons un menu escamotable : lorsqu'on lit le texte sans approcher le pointeur de la souris de la zone de menu, celui-ci est retracté, limité à une zone que l'on peut identifier et survoler avec la souris. Quand cette zone est survolée, l'élément en mode `hover`, modifie son attribut `clip` : en passant à la valeur `auto`, les dimensions de l'élément sont attribuées et l'élément semble surgir d'une boîte.

La réalisation pratique passe ici par une liste non ordonnée `ul` dont on n'affiche pas la puce. La zone a une bordure extérieure nulle et un padding nul excepté le padding de gauche qui est mis à `25px` : ce sera la largeur du rectangle de découpage du `clip`. On a limité la dimension verticale de ce rectangle à la hauteur un item `li` de la liste, soit `4em`.

<pre> lien 1 lien 2 lien 3 </pre>	<pre>ul{position: absolute; margin : 0; width : 150px; padding : 0 0 0 25px; border : solid 1px #0000; clip :rect(1px 25px 4em 1px); font :1em/3 Verdana sans-serif; list-style-type : none; text-transform : uppercase; background-color: #142857;} li{line-height : 4em; text-align : center;} a {display : block; height : 4em; text-decoration : none; background-color: #eee; color : #142857;} ul:hover{clip : auto;} a:hover{color : #eee; background-color: #142857;}</pre>						
	<table border="1"> <thead> <tr> <th data-bbox="592 383 847 412">non survol de la bordure</th> <th data-bbox="847 383 1086 412">survol de la bordure</th> <th data-bbox="1086 383 1326 412">survol d'un lien</th> </tr> </thead> <tbody> <tr> <td data-bbox="592 412 847 636"></td> <td data-bbox="847 412 1086 636"></td> <td data-bbox="1086 412 1326 636"></td> </tr> </tbody> </table>	non survol de la bordure	survol de la bordure	survol d'un lien			
non survol de la bordure	survol de la bordure	survol d'un lien					
							

L'effet obtenu est intéressant, mais il rester à inclure cet élément de menu dans une boîte, fixée à la page pour que le menu soit toujours accessible. Il reste aussi à définir comment le contenu de la page doit se comporter par rapport à ce menu, dans les deux aspects que celui-ci peut avoir.

La proposition qui suit ajoute seulement des blocs `div` différents pour le menu et le texte. On peut alors séparer la zone de menu de la zone de texte, le menu restant accessible à tout moment par le fait que sa zone est fixée à la fenêtre.

<pre>div.text{margin-left : 180px;} div.menu{position : fixed; margin : 0; font-align : center;}</pre>	 <p> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec</p>
<pre><div class="menu"> lien 1 lien 2 lien 3 </div> <div class="text"> <p>Lorem ipsum dolor sit amet, conse <p>Ut velit mauris, egestas sed, gra <p>Aliquam convallis sollicitudin p </div></pre>	 <p> Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit.</p> <p> Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus</p>

Propriétés des listes

Une liste à puces `ul` peut être mise en forme de deux façons :

- ✦ avec un attribut global définissant son type `type`, qui peut être `none` (pas de puce), `circle` (cercle vide), `disc` (cercle plein, par défaut) ou `square` (carré plein).
- ✦ avec l'attribut spécifique `list-style-type` qui peut prendre les valeurs globales (`none`, `circle`, `disc` ou `square`) ou de très nombreuses autres :
 - un code unicode⁹ comme `list-style-type: "\4DD1"` (hexagramme du travail sur le déclin ☹️) ou `"\A9"` (symbole copyright ©).
 - une image `puce.png` stockée dans un dossier `images` :

```
list-style-image: url("images/puce.png")
```

De même, on peut agir sur une liste numérotée `ol` :

- ✦ avec un attribut global définissant son type `type` (déjà été mentionné plus haut), qui peut être `1` (numérotation arabe, par défaut), `I` (numérotation romaine), `A` (alphabétique majuscule), `a` (alphabétique minuscule) et `i` (romain littéral).
- ✦ avec l'attribut spécifique `list-style-type` dont les valeurs possibles sont nombreuses, citons `decimal-leading-zero` (01, 02, etc.) et `lower-greek` (α , β , etc.).

9. Chercher ces codes unicode sur http://www.babelstone.co.uk/Unicode/unicode_fr.html

3. Programmation Javascript

Javascript est un langage de programmation adapté au HTML qui permet de traiter des événements propres à la navigation (pression sur un bouton de formulaire, mouvement ou clic de souris, etc.) et d'enrichir une page WEB (adjonction d'une date, gestion d'un compteur de fréquentation, etc.) ; il peut aussi servir à mesurer une audience (origine de la visite, durée, fidélité, etc.) ou à contrôler les données saisies d'un formulaire.

Le code Javascript est interprété directement sur la machine du client (la page WEB a été délivrée par un serveur pour un client), au niveau de son navigateur. Comme les codes HTML et CSS, le code Javascript d'une page WEB est accessible lorsqu'on demande d'accéder au code source de la page. Il ne faut pas confondre Javascript et Java qui est un langage compilé, ayant une syntaxe très différente de Javascript, qui n'a pas pour vocation d'être utilisé par une page WEB au niveau du client.

Pour des raisons de sécurité, Javascript ne peut pas lire les fichiers de l'utilisateur ni récupérer les mots de passe ou les e-mails. Les seules façons qu'il a de récupérer ce genre d'information est de les demander sous la forme d'un formulaire. Dans le même ordre d'idée, un code Javascript ne peut pas installer un programme exécutable, qu'il soit légitime (logiciel gratuit, antivirus) ou malveillant (spy-ware ou logiciel de prise de contrôle). D'autres composants que Javascript autorisent ces fonctionnalités (ActiveX ou les applets Java) ce qui oblige l'utilisateur du WEB à la plus grande prudence.

D'une façon générale, il y a de bonnes pratiques à mettre en œuvre, dès qu'il s'agit de programmation :

- ♦ l'indentation n'est pas obligatoire en Javascript (contrairement à Python), mais elle permet de rendre le code lisible pour une meilleure compréhension. De même, il n'est pas obligatoire d'utiliser le point-virgule ; à la fin d'une instruction. Mais cette pratique, obligatoire dans d'autres langages, est souvent conseillée.
- ♦ les commentaires ne sont jamais obligatoires, mais ils aident également la compréhension, y compris celle de l'auteur du code qui peut avoir du mal à se relire après quelque temps. En Javascript, les commentaires s'écrivent `//...` en fin de ligne ou bien `/* ...*/` sur plusieurs lignes.
- ♦ les règles de nommage ne sont pas universelles, mais il vaut mieux une variable portant un nom explicite comme `nbr_visites` plutôt qu'une variable nommée `v`. La séparation des mots se fera de préférence avec une majuscule plutôt qu'un *underscore*, ainsi `prixTotalTTC` est préférable à `prix_total_TTC`, bien que cela se discute. Utiliser toujours la même façon de nommer, en particulier les fonctions d'accès `get...()`, de modification `set...()`, d'ajout `add...()` ou d'identification `is...()`. On utilise le *s* du pluriel pour les tableaux alors que le singulier est utilisé pour les variables (`jours` est un tableau alors que `jour` est une variable).

Du code Javascript peut être inséré entre les balises

`<script type="text/javascript">... </script>`, à n'importe quel endroit d'un fichier HTML. Ainsi, l'instruction `<script "text/javascript">document.write("coucou !")</script>` placée dans le texte d'une page écrira simplement « coucou ! » dans cette page, tandis que `<script "text/javascript">console.log("coucou !")</script>` l'écrira dans la console WEB (accessible par le menu du navigateur).

Voici un exemple de code Javascript qui peut avoir son utilité dans une page WEB. Il montre qu'on peut intégrer du code HTML dans le code Javascript (la balise `
` qui crée un saut de ligne). Il montre également la déclaration d'une variable (mot-clé `var`) ainsi que l'utilisation d'une fonction prédéfinie (fonction `Date()` qui contient des champs accessibles par des fonctions spécifiques).

Date du jour
7/8/2019

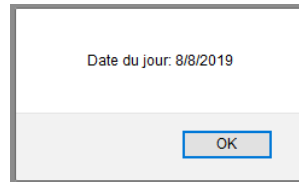
```
<script type="text/javascript">
var ladate=new Date();
document.write("Date du jour<br>");
document.write(ladate.getDate()+"/"+(ladate.getMonth()+1)+"/"+ladate.getFullYear());
</script>
```

La fonction Javascript `alert()` est intéressante, notamment en phase de test, puisqu'elle déclenche l'ouverture d'une fenêtre contenant un message, qui doit être fermée pour continuer les opérations. Cette méthode évite d'écrire dans la page (information hors contexte) et n'utilise pas la console (pas toujours facile d'accès).

Pour reprendre l'exemple précédent, on écrirait le message ainsi :

```
alert("Date: "+ladate.getDate()+"/"+(ladate.getMonth()+1)+"/"+ladate.getFullYear());
```

La fenêtre s'affiche alors ainsi (cela dépend du navigateur) :



On verra qu'il est possible d'intégrer un code Javascript à un élément HTML.

L'élément `form` qui déclare un formulaire peut recevoir des instructions Javascript par la syntaxe `<form action="javascript:(function(){...})()">`

Si on utilise un *eventHandler* (gestionnaire d'évènement), on peut par exemple écrire

```
<baliseName eventHandler="javascript:(function(){...})()">
```

L'autre méthode pour introduire du code Javascript est le fichier externe :

Le fichier doit être enregistré avec l'extension `js` et lié au document dans l'entête de celui-ci, par la déclaration HTML : `<script type="text/javascript" src="nameFichier.js" />`. Ce principe, déjà utilisé pour le code CSS, permet d'écrire du code Javascript utilisable par plusieurs pages HTML.

Langage Javascript

En Javascript, les variables n'ont pas de type prédéfini (pas de différence pour un entier, une chaîne de caractères, une liste) mais il en existe de deux types :

- ♦ les variables à portée *locale* sont déclarées à l'intérieur d'une fonction et avec le mot-clé `var`. Elles sont visibles uniquement dans la fonction où elles sont déclarées
- ♦ les variables à portée *globale* sont déclarées à l'extérieur d'une fonction ou sans le mot-clé `var`. Elles sont visibles partout.

Un fois qu'on a affecté une valeur à une variable, celle-ci possède un type auquel on peut accéder en tapant `typeof(varName)`. Pour les nombres (entiers, flottants, petits ou grands), il n'y a qu'un seul type : `number`. Il existe le type `string` (chaîne de caractères), `boolean` (booléen), `undefined` (déclaré mais pas encore affecté, ne peut être utilisé) ou encore `Array` (tableau).

Une fonction est un bloc d'instructions (limité par des accolades `{...}`) acceptant une liste de paramètres (il peut n'y en avoir aucun) et défini par le mot-clé `function`. Elle peut posséder un nom et retourner éventuellement une valeur (grâce à `return ...`).

Voici un syntaxe complète et correcte :

```
function nameFonction(param1, param2, ...){instr1;instr2; ...; return varia;}
```

La fonction suivante permet d'ajouter un `s` lorsqu'une variable `quantite` est supérieure à 1 :

```
function getS(quantite){if(quantite>1){return "s";}}
```

La voici utilisée dans un script qui écrira *Vous avez commandé 5 articles* :

```
var nbrArticle=5;
```

```
document.write("Vous avez commandé ",nbrArticle," article",getS(nbrArticle));
```

En mettant la variable `var nbrArticle` à 1, on obtiendrait *Vous avez commandé 1 article*.

Bien sûr cette fonction convient pour le mot *article*, mais ne conviendrait pas pour les mots *colis* (invariable) ou *cheval* (pluriel en x). Cette fonction peut donc être amenée à évoluer, selon les besoin.

Les instructions conditionnelles sont écrites selon la syntaxe

```
if(condition){instructions} else {instructions}
```

où les instructions sont enfermées dans un bloc limité par des accolades `{...}`, comme pour les fonctions. S'il n'y a qu'une seule instruction, on

peut enlever les accolades. La partie `else {instructions}` est facultative : on ne l'écrit que s'il y a des instructions à exécuter lorsque la condition n'est pas satisfaite. Une condition est toujours entourée de parenthèses. Ce peut être :

- ♦ un test d'égalité (noté avec `==`, comme en Python), de différence (noté avec `!=`) ou d'inégalité (notés classiquement `<`, `>`, `<=` ou `>=`). Il existe en Javascript un test d'égalité stricte qui s'écrit `===` (les variables doivent alors être de même type pour être égales).
- ♦ un expression booléenne utilisant les mot-clés `||` (le *ou* logique) ou `&&` (le *et* logique) entre plusieurs expressions booléennes.

Notons qu'une expression contenant `false && conditionX` sera toujours `false`, la `conditionX` n'étant pas examinée. De même, `true || conditionY` sera toujours `true`, la `conditionY` n'étant pas examinée. L'ordre d'écriture d'une condition composée a donc une importance.

Une syntaxe intéressante lorsqu'une variable peut prendre un nombre limité de valeurs : le mot-clé `switch` permet de donner des instructions pour chaque cas.

Dans le cas où les différentes valeurs possibles de la variable `nameVar` sont 1, 2, ..., la syntaxe est :

```
switch(nameVar){
  case(1):instructions;break;
  case(2):instructions;break;
  ...
  default:instructions;}
```

Le mot-clé `break` permet ici de stopper la lecture des instructions liées à un `case` et d'aller après l'accolade fermante `}`. Si aucun des cas envisagés par les `case` n'est trouvé, ce sont les instructions écrites après le mot-clé `default` qui sont exécutées.

La boucle `for` se programme en Javascript avec la syntaxe

`for(initiation;arrêt;incrément){instructions}` où les parties situées entre parenthèses ont des fonctions bien définies :

- ♦ `initiation` correspond à ce qui doit être exécuté avant d'entrer dans la boucle. Il s'agit généralement d'initialiser le compteur de boucle, typiquement `var i=0`.
- ♦ `arrêt` correspond au test d'exécution sur le compteur de boucle. Typiquement, si on veut que le compteur aille jusqu'à 100, on écrira `i<=100`.
- ♦ `incrément` indique comment est manipulé le compteur. Normalement, on ajoute 1 à chaque passage, on écrit donc `i=i+1` ou `i+=1` ou `i++` qui ont la même signification. On peut incrémenter la variable de boucle d'autre chose que de 1 (par exemple de 2 ou de 0.1).

Une boucle peut toujours être arrêtée avec le mot-clé `break` (éventuellement précédé d'un test) qui, dans le cas de boucles imbriquées, ne fait sortir que de la boucle intérieure (celle qui contient l'instruction). Il est possible aussi d'utiliser le mot-clé `continue` qui fait passer au tour suivant sans examiner les instructions qui suivent ce mot-clé.

Une boucle conditionnelle se programme en Javascript avec la syntaxe

`while(condition){instructions}`. Le passage dans la boucle s'effectue tant que la condition écrite dans `condition` est satisfaite. On peut utiliser une autre syntaxe qui oblige à passer au moins une fois dans la boucle : `{do{instructions}while(condition)}`; cette syntaxe `do ... while` est toutefois moins courante que la classique boucle `while`.

Les tableaux en Javascript sont des objets de la classe `Array` (cela explique le A majuscule).

Un objet est créé par un constructeur : pour déclarer un tableau `tab` vide, on écrit `tab=new Array()`. On peut aussi déclarer et initialiser un tableau avec les valeurs qu'il contient, comme par exemple `tab=new Array(1,2,3,4,5)` ou `tab=new Array("oui","non")` ou encore même, sans le constructeur, comme en Python `tab=[1,2,3,4,5]`.

Pour accéder à un élément d'un tableau, on utilise les crochets, comme en Python : `var a=tab[0]`, l'indice du 1^{er} élément étant 0. Cette notation permet également de modifier la valeur stockée. On pourra, par exemple, écrire `tab[1]="peut-être"` qui mettra dans `tab[1]` la valeur indiquée. La syntaxe est très souple, comme souvent avec les applications WEB : on peut déclarer un tableau et initialiser juste une valeur `tab=new Array(); tab[5]=12;`, cela ne pose pas de problème. Si vous demander alors la valeur de `tab[5]`, c'est bien 12, tandis que celles de `tab[0]` ou `tab[47]` sont tout simplement `undefined`.

La propriété `length` de l'objet `Array` donne la longueur du tableau (son nombre d'éléments). On obtient la longueur du tableau `tab` en écrivant `tab.length` (notation pointée, réservée aux objets). Pour recopier un tableau, il ne faut pas procéder par une simple affectation. Si `tab1` est un tableau, la déclaration `var tab2=tab1` conduirait à un tableau inutilisable : modifier une valeur de `tab2` modifierait également la valeur correspondante dans `tab1`. Pour que la copie soit indépendante du modèle, il faut déclarer un nouveau tableau (avec le constructeur `new`) et recopier le tableau initial élément par élément. Voici une syntaxe correcte pour effectuer cela :

```
tab2=new Array();
for(var i=0;i<tab1.length;i++){tab2[i]=tab1[i];}
```

Un objet Javascript est une structure hiérarchisée de données regroupées sous une seule entité, qui facilite sa manipulation. En réalité presque toutes les données Javascript manipulées (les données du document, du navigateur, les tableaux, les chaînes de caractères, etc.) sont des objets. Un objet peut avoir des *propriétés* et des *méthodes* :

- ♦ une propriété est une donnée liée à l'objet. Le titre de la page du document, par exemple, est connu sous le nom `document.title` et peut être lu ou modifié : `title` est une des propriétés de l'objet `document`.
- ♦ une méthode est une fonction. Pour écrire sur la page du document, on a donné la fonction `document.write()` : `write` est une des méthodes de l'objet `document`. La méthode `alert` mentionnée plus haut appartient à l'objet `window` (fenêtre du navigateur), mais on peut se passer du préfixe pour cet objet. D'ailleurs `document` est lui-même une propriété de `window`...

Pour créer un nouvel objet, on le déclare en écrivant un constructeur. Un constructeur est une fonction qui structure les données entrées en argument, en leur attribuant des noms de propriété et qui en déclare les méthodes. Pour utiliser un objet, on l'instancie en appelant le constructeur au moyen du mot-clé `new` comme cela a été vu.

EXEMPLE 3 – Créons l'objet `Repertoire` contenant le nom et la date de naissance de personnes, et calculant l'âge de ces personnes. Dans le constructeur de cet objet, on utilise le mot-clé `this` permettant de faire référence à l'objet qui a appelé cette fonction (ici l'objet en cours de création).

```
function Repertoire(txtNom, txtDate){
    this.nom=txtNom;.
    this.date=txtDate;.
    this.age=calculAge;}

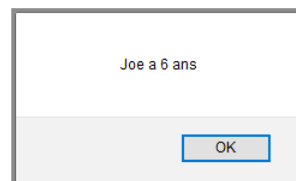
```

Ce constructeur permet d'instancier une variable `ami` contenant les données de Joe, né le 12 décembre 2012, avec la syntaxe `var ami=new Repertoire("Joe", "12/12/2012")`. Pour obtenir son nom, on doit taper `ami.nom` et pour obtenir son âge avec la méthode `ami.age()`, il faut encore créer la fonction `calculAge` : la date de naissance étant entrée au format "jj/mm/aaaa", on peut écrire la fonction ci-dessous.

```
function calculAge() {
    var txtDate = this.date.split("/");
    var birthMonth = txtDate[1]-1, // (les mois commencent à 0)
        birthDay = txtDate[0],
        now = new Date(),
        nowMonth = now.getMonth(),
        nowDay = now.getDate(),
        age = now.getFullYear()-txtDate[2];
    // Si la date d'anniversaire n'est pas encore passée, on corrige l'âge
    if(nowMonth<birthMonth || nowMonth==birthMonth && nowDay<birthDay) age--;
    return age;}

function Repertoire(txtNom, txtDate){
    this.nom=txtNom;
    this.date=txtDate;
    this.age=calculAge;}

var ami=new Repertoire("Joe","12/12/2012");
alert(ami.nom+" a "+ami.age()+" ans");
```



En plus des fonctionnalités déjà mentionnées, cette fonction utilise la méthode `split` de l'objet `string` qui transforme une chaîne de caractère en tableau, le séparateur étant le caractère entré en argument (ici le `/`).

Accès aux éléments HTML

Jusqu'à présent, nous n'avons mentionné que peu de possibilités d'interaction avec une page HTML (l'accès à la méthode `document.write` ou à la propriété `document.title`). Javascript peut, en réalité accéder à tous les éléments du code HTML. Il peut les lire et les modifier. Il peut aussi ajouter du nouveau code HTML. Pour cela, il dispose de plusieurs moyens.

* La propriété `getElementById` de l'objet `document`, qui s'emploie en écrivant `document.getElementById(idName)`, renvoie l'objet correspondant à l'id spécifié par `idName`. L'attribut `id` désigne un élément du code HTML : il a été entré dans le code HTML pour permettre son utilisation ultérieure par Javascript.

Si on a entré dans le code HTML l'élément `<div id="salutation">Madame</div>` et que l'on veut changer (d'après l'information sur le sexe de la personne) le texte englobé par cette balise en `Monsieur`, il faut retrouver l'élément concerné et le modifier dynamiquement.

Pour cela, on modifie la propriété `innerHTML` de `getElementById` :

```
document.getElementById("salutation").innerHTML="Monsieur"
```

De cette façon, l'utilisateur masculin (identifié comme tel) ne voit jamais `Madame` mais `Monsieur`.

Ce principe est simple et a beaucoup d'applications. Cependant certains navigateurs n'acceptent pas `getElementById`. Aussi est-il prudent, lorsqu'on veut s'adresser à n'importe quel utilisateur, de disposer un test de compatibilité avant l'utilisation de cette propriété. On écrira donc notre instruction entre accolades : `if(document.getElementById){...}`.

EXEMPLE 4 – Nous voulons écrire l'heure dans une page WEB. Pour cela, il faut utiliser l'objet `Date` contenant les méthodes `getHours()`, `getMinutes()` et `getSeconds()` qui donnent l'heure, les minutes et les secondes de l'instant présent. On disposera donc un appel à une fonction Javascript dans l'élément `body` du code HTML qui réalisera l'inscription de la balise

```
<div id="horloge">...</div>
```

dans laquelle seront inscrits les trois nombres dans le format `hh:mm:ss`. Au début, la balise ne contient pas l'heure, mais celle-ci est actualisée par une autre fonction qui modifie le texte englobé dans la balise `div`. Une fonction Javascript, très utile pour actualiser une information, est appelée pour changer cet affichage chaque seconde : la fonction `setTimeout("nameFonction()",1000)`.

<pre><script type="text/javascript"> function afficheHorloge(){ if(document.getElementById){var txtHorloge="<div id='horloge'> no time </div>"; document.write(txtHorloge); miseAJourHorloge();}} function miseAJourHorloge(){var dt=new Date(); var h=dt.getHours(), m=dt.getMinutes(), s=dt.getSeconds(); if(h<10) h="0"+h; if(m<10) m="0"+m; if(s<10) s="0"+s; document.getElementById("horloge").innerHTML=h+":"+m+":"+s; setTimeout("miseAJourHorloge()",1000);} </script> <style type="text/css"> #horloge{width:100px; border:2px solid #000; font-size:14px; font-weight:bold; background-color:#eee; text-align:center;} </style></pre>	
<pre>code HTML (dans body) <script type="text/javascript"> afficheHorloge(); </script></pre>	<pre>affichage dans la page</pre>

Remarquer l'utilisation des deux marqueurs d'une chaîne de caractères : `"...','...','..."` est utilisé ici pour adjoindre un texte à l'intérieur d'un texte. On pourrait aussi, comme en Python, utiliser la syntaxe d'échappement `"...\""...\""..."` qui est moins agréable à lire.

Le code des fonctions Javascript peut être écrit dans l'entête `head` du fichier HTML (c'est l'option retenue ici) ; il pourrait aussi être écrit dans un fichier séparé lié comme on l'a dit au fichier HTML. Cela est d'autant plus vrai que l'heure est une information que l'on peut souhaiter lire dans d'autres pages. La même chose peut être dite du code CSS que nous avons ajouté pour obtenir un meilleur affichage de l'heure.

* La méthode `getElementsByName` de l'objet `document`, qui s'emploie en écrivant `document.getElementsByName(nameName)`, renvoie *un tableau* contenant les objets correspondant à l'attribut `name` spécifié par `nameName`. Remarquer le **s** mis à `Element` qui indique le pluriel. La spécification de l'attribut `id` utilisé par la méthode précédente est qu'il ne doit être employé que pour un unique élément ; pour l'attribut `name`, il n'en est rien. Aussi, dès que l'on veut appliquer un traitement à plusieurs éléments (par forcément de même nature), on peut employer cette dernière méthode. Signalons deux autres méthodes qui fonctionnent sur le même principe : `getElementsByTagName` et `getElementsByClassName` : la 1^{re} permet de retrouver les éléments à partir de leur `tag` (le nom de balise) et la 2^e permet de retrouver les éléments à partir de leur attribut `class`.

EXEMPLE 5 – Les propriétés de style, le code CSS, peut être modifié par Javascript. Voici un exemple qui montre comment l'attribut `display` de plusieurs éléments `div` ayant la même valeur de l'attribut `name` peut dynamiquement être modifié. En le faisant passer de la valeur `none` à la valeur `block`, on fait apparaître d'un coup dans la page WEB tous ces éléments.

Pour cet exemple, on va afficher une liste de différents sites apportant des précisions sur cette fonctionnalité de Javascript. La liste initiale tient en trois lignes qui donnent juste les noms de ces sites. En cliquant sur le lien « des détails supplémentaires », on obtient l'affichage des détails et le texte du lien propose de cacher les détails.

```
<script type="text/javascript">
var nbrClic=0;
function changeDisplay(){var x = document.getElementsByName("details");
    var i,value;
    nbrClic++;
    if(nbrClic%2==0){value="none";
        document.getElementById("question").innerHTML="des détails supplémentaires";}
    else{value="block";
        document.getElementById("question").innerHTML="cacher les détails";}
    for (i=0; i<x.length; i++) {x[i].style.display=value;}}
</script>
```

```
<style type="text/css">
div{width:400px; border:1px solid #000;
font-size:14px; font-weight:light;
background-color:#eee;}
</style>
```

```
<p>Voici quelques explications sur getElementsByName: </p>
<ul><li>w3schools.com
    <div name="details" style="display:none">
The getElementsByName() method returns a collection of all elements in the document with the specified name (the value of the name attribute), as a NodeList object.
<a href="https://www.w3schools.com/jsref/met_doc_getelementsbyname.asp" target="_blank">Ouvrir la page</a></div></li>
<li>developer.mozilla.org
    <div name="details" style="display:none">
Renvoie une liste des éléments portant un name donné dans le document (X)HTML.
<a href="https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByName" target="_blank">Ouvrir la page</a></div></li>
<li>toutjavascript.com
    <div name="details" style="display:none">
Retourne un tableau d'éléments HTML HTMLElement ayant nom défini dans la propriété name de la balise de l'objet.
<a href="https://www.toutjavascript.com/reference/ref-window.document.getelementsbyname.php" target="_blank">Ouvrir la page</a></div></li></ul>
<p>Voulez-vous <a href="javascript:changeDisplay()" id="question">des détails supplémentaires</a>?</p>
```

Voici quelques explications sur `getElementsByName`:

- w3schools.com
- developer.mozilla.org
- toutjavascript.com

Voulez-vous [des détails supplémentaires?](#)

Voici quelques explications sur `getElementsByName`:

- w3schools.com
The getElementsByName() method returns a collection of all elements in the document with the specified name (the value of the name attribute), as a NodeList object. [Ouvrir la page](#)
- developer.mozilla.org
Renvoie une liste des éléments portant un name donné dans le document (X)HTML. [Ouvrir la page](#)
- toutjavascript.com
Retourne un tableau d'éléments HTML HTMLElement ayant nom défini dans la propriété name de la balise de l'objet. [Ouvrir la page](#)

Voulez-vous [cacher les détails?](#)

Pour simplifier, j'ai inclus ici le déclenchement de la fonction Javascript `changeDisplay()` dans un lien. Pour appeler cette fonction, il faut donner à l'attribut `href` du lien la valeur `javascript:changeDisplay()`. Ce lien doit avoir son propre texte modifié lorsque les détails sont affichés. Pour effectuer cette modification depuis le script, un `id` est donné au lien.

La variable globale `nbrClic`, initialisée à 0, permet de passer d'un état à l'autre : à chaque passage dans `changeDisplay()` elle augmente de 1 et l'affichage des détails ne s'effectue que lorsqu'elle est impaire.

4. Dynamisme évènementiel

Il y a principalement un seul évènement en pur HTML : le clic sur un lien qui permet de renvoyer sur une autre page Web. Javascript ajoute de nombreux autres évènements et, en permettant de les associer aux autres fonctionnalités, augmente l'interactivité des pages WEB.

Ainsi, certains évènements peuvent être détectés par un élément HTML. On va voir ci-dessous comment une balise ``, par exemple, peut devenir sensible à un clic de souris et déclencher un code JavaScript lorsque l'utilisateur clique dans la zone délimitée par cet élément. Ceci dit, voici les principaux évènements et les actions à effectuer pour qu'ils se déclenchent.

Évènement	Action
click	Cliquer (appuyer puis relâcher) sur l'élément
dblclick	Double-cliquer sur l'élément
mouseover	Faire entrer le curseur sur l'élément
mouseout	Faire sortir le curseur de l'élément
mousedown	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
mouseup	Relâcher le bouton gauche de la souris sur l'élément
mousemove	Faire déplacer le curseur sur l'élément
keydown	Appuyer (sans relâcher) sur une touche de clavier sur l'élément
keyup	Relâcher une touche de clavier sur l'élément
keypress	Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
focus	« Cibler » l'élément (un élément ciblé va recevoir tous les évènements du clavier)
blur	Annuler le « ciblage » de l'élément (en utilisant la touche de tabulation par exemple)
change	Changer la valeur d'un élément spécifique aux formulaires (<code>input</code> , <code>checkbox</code> , etc.)
input	Taper un caractère dans un champ de texte (géré selon le navigateur)
select	Sélectionner le contenu d'un champ de texte (<code>input</code> , <code>textarea</code> , etc.)
submit	Envoyer (soumettre) le formulaire
reset	Réinitialiser les champs du formulaire

Le dynamisme évènementiel s'est mis en place progressivement et il subsiste différentes techniques pour réaliser une même action. voici, par exemple, trois façons de gérer l'évènement `click` sur un élément `` :

1. Sans le DOM¹⁰

Exemple : `Cliquez-moi !`

Il suffit de cliquer sur le texte pour que la boîte de dialogue s'affiche. Afin d'obtenir ce résultat nous ajoutons au `` un attribut formé du mot « on » suivi du nom de l'évènement « click » ce qui donne « onclick » (on peut noter cet attribut `onclick` ou `onClick` ou `ONCLICK`, les attributs HTML n'étant pas sensibles à la casse). Cet attribut possède une valeur qui est un code JavaScript : on peut y écrire quasiment tout ce qu'on souhaite, mais tout doit tenir entre les guillemets de l'attribut. En utilisant le mot-clé `this`, on peut intervenir sur l'élément HTML qui est à l'origine de l'évènement :

```
<p onclick="this.textContent='Merci !';">Cliquez-moi !</p>
```

On peut changer le code HTML : `this.innerHTML = 'Bravo !'`;

Ou bien le style : `this.style.color = 'orange'`;

2. Avec le DOM (version 0, datant de 1995) :

Une propriété gestionnaire d'évènement est affectée à un élément HTML.

```
<span id="clickme">Cliquez-moi !</span>
```

```
<script>var element=document.getElementById('clickme');
```

```
element.onclick=function(){alert("Vous m'avez cliqué !");};</script>
```

L'évènement est défini ici non plus dans le code HTML mais directement en JavaScript. Chaque évènement standard possède donc une propriété dont le nom est, à nouveau, précédé par le mot « on ». Cette propriété ne prend plus pour valeur un code JavaScript brut, mais une fonction (nommée ou anonyme). Ici, une fonction anonyme a été assignée à notre évènement. On aurait

10. Le DOM (*Document Object Model*) est le schéma qui décrit comment les éléments d'une page sont structurés.

pu créer une fonction avec un nom puis n'assigner que le nom de la fonction à notre évènement en écrivant :

```
function message(){alert("Vous m'avez cliqué !");};element.onclick=message;
```

3. Avec le DOM (version 2, datant de 2003) :

La méthode `addEventListener()` est affectée à un élément HTML.

```
<span id="clickme">Cliquez-moi !</span>
<script>var element=document.getElementById('clickme');
element.addEventListener('click',function(){alert("Vous m'avez cliqué !");});</script>
```

On n'utilise plus une propriété mais une méthode, nommée `addEventListener()`, et qui prend trois paramètres (les deux premiers seulement dans l'exemple) :

- (a) Le nom de l'évènement (sans le mot « on »)
- (b) La fonction à exécuter
- (c) Un booléen optionnel pour spécifier si l'on souhaite utiliser la phase de capture (début d'évènement : `true`) ou bien celle de bouillonnement (par défaut, fin d'évènement : `false`)

On aurait pu aussi déclarer la fonction comme une variable :

```
<script>var element=document.getElementById('clickme');
var message=function(){alert("Vous m'avez cliqué !");};
element.addEventListener('click',message);</script>
```

L'avantage de cette méthode est qu'elle peut être répétée pour un même élément, afin de gérer plusieurs types d'évènements (cela n'est pas possible avec la propriété du DOM-0) :

```
element.addEventListener('mouseover',Fonction1);
element.addEventListener('mousedown',Fonction2);
function Fonction1(){this.innerHTML = 'Cliquez moi !';
this.style.backgroundColor='orange';};
function Fonction2(){this.innerHTML = 'Bravo !';
this.style.color='#26C';this.style.fontWeight='bold';this.style.fontSize='24px';};
```

Si on veut gérer l'ensemble des actions de la souris, il faut associer une fonction à quatre évènements : `mouseover` (la souris passe sur l'élément), `mouseout` (la souris sort de l'élément), `mousedown` (la souris est cliquée sur élément) et `mouseup` (le bouton de la souris est relâché de l'élément).

Tous les éléments ne supportent pas tous les types d'évènements¹¹ et tous les navigateurs ne gèrent pas de la même façon les évènements. Sans entrer davantage dans les complications, examinons quelques-unes des très nombreuses possibilités apportées par ce dynamisme évènementiel.

L'objet Event

L'objet `Event` apporte une multitude d'informations sur l'évènement déclenché : on peut récupérer les touches actuellement enfoncées, les coordonnées du curseur, l'élément qui a déclenché l'évènement, etc. Pour récupérer une référence vers l'objet `Event` qui a été détecté, on met un argument à la fonction gestionnaire de l'évènement (cet argument est noté ici `event`, mais on peut le nommer comme on veut) :

```
element.onclick = function(event) {alert(event.type);};
element.addEventListener('click', function(event) {alert(event.type);});
```

La propriété `type` affiche le type de l'évènement (`click`, `mouseover`, etc.).

À la place de cette `alert`, on aurait pu changer le code HTML avec

```
event.target.innerHTML = 'Vous avez cliqué !'.
```

La propriété `target` permet de récupérer une référence vers l'élément dont l'évènement a été déclenché (exactement comme un `this` le ferait).

La position du curseur est une information très importante (pour faire du *drag & drop*).

Généralement, on récupère la position du curseur par rapport au coin supérieur gauche de la page Web. Celle-ci est alors connue par deux propriétés : `clientX` pour la position horizontale et `clientY`

11. Consulter par exemple la liste <https://www.commentcamarche.net/contents/573-javascript-les-evenements>

pour la position verticale. L'évènement le plus adapté à la majorité des cas est `mousemove` (la souris bouge). Pour récupérer la position du curseur par rapport à l'écran, utiliser `screenX` et `screenY`.

```
<div id="position"></div>
```

```
<script>var position = document.getElementById('position');
```

```
document.addEventListener('mousemove',function(e){position.innerHTML=
```

```
'Position X : ' + e.clientX+'px<br />Position Y : ' + e.clientY + 'px';});</script>
```

Ce script récupère et affiche les coordonnées du curseur dans la page WEB.

La récupération des touches frappées se fait par le biais de trois évènements différents. Les évènements `keyup` et `keydown` sont conçus pour capter toutes les frappes de touches. Ainsi, il est parfaitement possible de détecter l'appui sur la touche A, voire même sur la touche `Ctrl`. La différence entre ces deux évènements se situe dans l'ordre de déclenchement : `keyup` se déclenche au relâchement de la touche, tandis que `keydown` se déclenche au moment de l'appui sur la touche (comme `mousedown`). Attention : toutes les touches retournant un caractère le retourne en majuscule (que la touche `Maj` soit pressée ou non). L'évènement `keypress` ne détecte que les touches qui écrivent un caractère (pas les touches `Ctrl` et `Alt` par exemple). On peut ainsi détecter les combinaisons de touches : avec la combinaison `Maj + A`, l'évènement `keypress` détecte A majuscule tandis que `keyup` ou `keydown` se déclenchent deux fois (une fois pour la touche `Maj` et une deuxième fois pour la touche A). Trois propriétés sont capables de fournir le code ASCII correspondant à la touche pressée : `keyCode`, `charCode` et `which`. La méthode `String.fromCharCode(c1,c2,...)` renvoie les caractères correspondants aux codes entrés, par exemple l'instruction `String.fromCharCode(84, 101, 115, 116)` affiche "Test".

EXEMPLE 6 – La fonction `prompt` est une variante de la fonction `alert` : elle affiche une boîte de dialogue où l'utilisateur peut saisir du texte. La méthode retourne le texte saisi si le visiteur clique sur `Valider` et retourne `null` si le visiteur clique sur `Annuler` ou appuie sur la touche `Echap`. La syntaxe est `prompt(message, défaut)` où `défaut` est le texte écrit par défaut dans la boîte. Faisons en sorte qu'en cliquant sur un item d'une liste ordonnée `` qui peut ne pas être la seule liste du document, une zone de texte `prompt()` s'affiche et propose de modifier le texte de l'item. Pour cela, identifions la liste concernée dans une `<div id="output">` et ajoutons un gestionnaire d'évènement `onclick` sur chaque item, susceptible de déclencher une fonction affichant le `prompt()` qui modifiera le texte. Dans la fonction lancée par un clic, on récupère le texte initial avec `this.innerHTML` et on l'affiche comme second paramètre du `prompt()`.

```
<div id="output">Quelques langages interprétés:
<ol><li>Ruby</li>
  <li>Python</li>
  <li>PHP</li>
  <li>Javascript</li></ol>
</div>
<script>var output=document.getElementById('output');
  if(output){var items=output.getElementsByTagName('li');
    for(var i=0,c=items.length;i<c;i++){
      items[i].onclick=function(){if(text=prompt("Modifiez le texte",this.innerHTML)){this.innerHTML=text;}};}}
</script>
<style type="text/css">div{width:200px; border:1px solid #000;font-size:14px; font-weight:light;background-color:#eee;}
</style>
```

Quelques langages interprétés:

1. Ruby
2. Python
3. PHP
4. Javascript

Modifiez le texte

OK Annuler

Modifiez le texte

OK Annuler

Quelques langages interprétés:

1. Perl
2. Python
3. PHP
4. Javascript

On aurait pu ici utiliser la méthode `addEventListener` :

```
for (var i=0, c=items.length; i<c; i++){items[i].addEventListener("click",
  function(){if(text=prompt("Modifiez le texte",this.innerHTML)){this.innerHTML=text;}},
  false);}
```

Ainsi, on peut modifier notre liste initiale (remplacer un langage interprété par un autre, par exemple Perl à la place de Ruby).

Citons pour finir la 3^e option de boîte de dialogue : la fonction `confirm` qui affiche le paramètre texte dans un message d’alerte avec les deux boutons de confirmation. Elle retourne `true` si le visiteur clique sur Ok et `false` si le visiteur clique sur Annuler ou appuie sur la touche Echap.

Formulaires

Un formulaire `<form name="form1">` est un élément de l’objet `document`.

Pour accéder à ce formulaire, on peut écrire : `document.forms["form1"]` (`forms` est un tableau contenant tous les formulaires du document) ou `document.form1` ou `document.forms[0]` (si `form1` est le 1^{er} formulaire du document). Je ne retiendrai plus cette dernière notation qui est gênante si on est amené à changer l’ordre des éléments).

Les champs de texte

Supposons que le formulaire contienne juste un champ de texte. Par exemple :

```
<form name="form1"><input type="text" name="champ1" value="Valeur initiale"></form>
```

Pour accéder à ce champ, on peut écrire : `document.forms["form1"].elements["champ1"]`

(`elements` est un tableau contenant tous les éléments du formulaire) ou `document.form1.champ1`.

une notation abrégée utilisant le mot-clé `this` est `this.form.champ1` (`this.form` est le formulaire de l’élément en cours). On peut ainsi modifier le contenu de la zone texte :

```
document.forms["form1"].elements["champ1"].value="Entrer votre texte ici".
```

On peut donner le focus à ce champs : `document.forms["form1"].elements["champ1"].focus();`

Pour vider un champ à la prise de focus, par exemple, un champ de saisie de texte (login) contenant à l’origine "Votre login" :

```
<input type="texte" name="login" value='Votre login'
  onfocus="if(this.value=='Votre login') {this.value=''}">
```

Remarque : on peut donner le contenu initial du champ avec l’attribut `placeholder`.

Cette indication disparaissant automatiquement lorsque le visiteur clique à l’intérieur du champ.

Les cases à cocher

Voici un formulaire contenant deux cases à cocher et un bouton.

```
<form><input type="checkbox" name="majeur">Majeur
<input type="checkbox" name="etudiant">Etudiant
<input type="button" value="Test" onclick=
  "console.log('Majeur: '+this.form.majeur.checked+
    '\n Etudiant: '+this.form.etudiant.checked);"></form>
```

La propriété `checked` d’une case à cocher est à `true` si la case a été cochée (à `false` sinon).

L’action du bouton Test est, ici, d’écrire sur la console l’état des deux cases.

Les radio-boutons

La gestion des radio-boutons est assez semblable à celle des cases à cocher (sauf qu’ici une seule case peut être cochée) : tous les radio-boutons étant liés par un même nom (une même valeur de l’attribut `name`), pour déterminer lequel est sélectionné il suffit de parcourir la liste des radio-boutons liés et de repérer lequel a la propriété `checked` à `true`.

Voici un exemple qui écrit le choix dans la console (par exemple « Sexe=f ») :

```
<form> Sexe :</br>
<input type="radio" name="sx" value="h" checked> homme
<input type="radio" name="sx" value="f"> femme
<input type="button" value="Test" onclick="testerRadio(this.form.sx)"></br></form>
<script type="text/javascript">function testerRadio(r)
  {for(var i=0;i<r.length;i++){if(r[i].checked){console.log("Sexe="+r[i].value)}}}</script>
```

Les sélecteurs à choix multiple

Voici un sélecteur à choix multiple (de type `select`) :

```
<form name="form1"><select name="os" size=1>
  <option value="val1">linux</option>
  <option value="val2">windows</option>
```

```
<option value="val3">ios</option></select></form>
```

Pour récupérer l'indice de la ligne sélectionnée : `this.form.elements['os'].selectedIndex`

Pour récupérer la valeur de la ligne sélectionnée (par exemple "val1") :

```
this.form.elements['os'].options[this.form.elements['os'].selectedIndex].value
```

(options est un tableau contenant les différentes lignes du sélecteur).

De même, pour récupérer le texte de la ligne sélectionnée (par exemple "linux") :

```
this.form.elements['os'].options[this.form.elements['os'].selectedIndex].text.
```

L'illustration ci-dessous montre cette récupération du texte et son affichage.

```
<form name="form1">
  <select name="os" size=1 onchange="testerSelect(this)">
    <option value="val1">linux</option>
    <option value="val2">windows</option>
    <option value="val3">ios</option>
  </select>
</form>
<p>Votre sélection : <span id="choix"></span></p>
<script>function testerSelect(s){document.getElementById("choix").innerHTML=s.options[s.selectedIndex].text};</script>
<style>#choix{width:200px; font-size:14px; font-weight:light;background-color:#eee;}</style>
```

Validation d'un formulaire

Pour valider un formulaire, on peut utiliser un bouton `submit` qui soumet directement le formulaire (ce bouton correspond à une balise `<input type="submit">`), mais il est souvent utile de vérifier la saisie d'un formulaire avant de le valider. L'idéal est de créer un bouton de type "button" (et pas "submit") qui appelle une fonction Javascript contrôlant la saisie et soumet (ou non) le formulaire.

Exemple : contrôle de saisie d'un email.

Une adresse mail contenant obligatoirement le caractère @, le contrôle se limite à cette vérification.

```
<script type="text/javascript">function ValiderMail(f){
  if (f.mail.value.indexOf("@",0)<0) {alert("Adresse email invalide")}
  else {if(confirm("Adresse validée.\nEnvoyer l'adresse?")){f.submit();}}}</script>
<form name="form1" action="">Saisissez votre adresse mail :
  <input type="texte" name="mail">
  <input type="button" name="bouton" value="Valider" onclick="ValiderMail(this.form)">
</form>
```

Sur le formulaire classique d'inscription ci-dessous, de nombreux points peuvent faire l'objet d'un contrôle : l'adresse mail et le mot de passe doivent avoir une syntaxe particulière et certains champs sont requis (ils ne peuvent rester vides). Pour avertir immédiatement l'utilisateur, les contrôles sont de préférence effectués en temps réels : le mot de passe par exemple est traité au moment de la saisie (événement `input`), l'adresse mail est contrôlée à la perte du focus (événement `blur`) ; l'extrait de fichier Javascript montre cela.

Formulaire d'inscription

Pseudo :

Mot de passe : Longueur : faible

Courriel : Adresse invalide

M'envoyer un courriel de confirmation

M'abonner à la newsletter et aux promotions

M'abonner uniquement à la newsletter

Ne pas m'abonner

Nationalité :

```
document.getElementById("mdp").addEventListener("input",
function(e){var mdp=e.target.value;
var longueurMdp="faible";
var couleurMsg="red";
if (mdp.length>8){longueurMdp="suffisante";couleurMsg="green";}
else if (mdp.length>=4){longueurMdp="moyenne";couleurMsg="orange";}
var aideMdpElt=document.getElementById("aideMdp");
aideMdpElt.textContent="Longueur : "+longueurMdp;
aideMdpElt.style.color=couleurMsg;});
```

```
document.getElementById("mail").addEventListener("blur",
function(e){var valMail="";
if(e.target.value.indexOf("@")===-1){valMail="Adresse invalide";}
document.getElementById("aideMail").textContent=valMail;});
```

Il existe en HTML5 de nouveaux types de balises `input` qui facilitent la validation d'un formulaire mais ceux-ci ne sont pas encore supportés par tous les navigateurs : les types `"url"` (pour les adresses URL) ou `"email"` (pour les adresse mail) par exemple sont entourés de rouge si les contenus ne sont pas valides dans le navigateur Firefox ; les types `"tel"` (pour un numéro de téléphone) ou `"url"` déclenche l'ouverture d'un clavier adapté sur certains navigateurs mobiles ; il existe aussi les types `"color"`, `"range"`, `"date"`, `"search"`, etc.

Pour les contrôles plus stricts des données d'un formulaire, il est fait usage des expressions régulières, une notion très performante mais assez compliquée que nous ne développerons pas ici ¹².

Soumission d'un formulaire

Les données saisies par l'utilisateur et contrôlées côté client (dans le navigateur) grâce à JavaScript sont ensuite envoyées via le réseau à un serveur WEB qui va les traiter et renvoyer une réponse adaptée au navigateur client sous la forme d'une nouvelle page WEB. Pour réaliser cette tâche, le serveur emploie une technologie adaptée, comme par exemple le langage PHP.



* Étape 1 : le client envoie une requête au serveur en utilisant le protocole HTTP.

Deux attributs de l'élément `<form>` définissent la méthode d'envoi des données :

- ✦ L'attribut `action` définit où les données sont envoyées : sa valeur est une URL valide (*Uniform Resource Locator*, une ressource identifiée par son emplacement). Si elle n'est pas fournie, les données sont envoyées à l'URL de la page contenant le formulaire. Anciennement, il fallait indiquer par `action="#"` que les données sont envoyées à l'URL de la page contenant le formulaire, en HTML5 ce n'est plus nécessaire. Il est possible de spécifier une URL qui utilise le protocole HTTPS (HTTP sécurisé). Quand vous faites ceci, les données sont chiffrées avec le reste de la requête, même si le formulaire lui-même est hébergé dans une page non sécurisée à laquelle on accède via HTTP.
- ✦ L'attribut `method` définit comment les données sont envoyées. Une requête HTTP consiste en deux parties : un en-tête (`header`) qui contient les métadonnées sur le navigateur, et un corps (`body`) qui contient les informations nécessaires au serveur pour effectuer la requête. Les données des formulaires HTML peuvent être envoyées via au moins deux méthodes :
 - La méthode `GET` envoie un corps vide, les données envoyées au serveur sont ajoutées à l'URL. En utilisant cette méthode, on voit apparaître dans l'URL, les champs du formulaire. Avec le formulaire


```
<form action="http://bidon.fr/traitement.php" method="get">
```

 par exemple, contenant les champs de texte `<input name="dire" value="Bonjour">` et `<input name="a" value="Maman">`, on va voir apparaître dans la barre du navigateur à la soumission du formulaire l'URL complétée `www.bidon.fr/?dire=Bonjour&a=Maman` : les données ont été ajoutées à l'URL après un point d'interrogation, sous la forme d'une suite de paires nom/valeur séparées par une esperluette (le caractère `&`).
 - La méthode `POST` envoie les données dans le corps de la requête (aucune donnée n'est ajoutée à l'URL qui s'inscrit dans la barre du navigateur). Si on a besoin d'envoyer un mot de passe ou toute autre donnée sensible, il ne faut jamais utiliser la méthode `GET` qui afficherait celui-ci en clair dans la barre d'URL. La méthode `POST` est également préférable quand on a besoin d'envoyer une grande quantité de données car certains navigateurs et de nombreux serveurs limitent la taille des URLs qu'ils acceptent (limitation à 255 caractères).

* Étape 2 : Le serveur traite la demande et renvoie une page au client.

À la soumission du formulaire, les données sont envoyées au fichier PHP du serveur indiqué par l'attribut `action`. Avec le formulaire précédent de balise

12. Voir à ce sujet https://www.w3schools.com/js/js_regexp.asp

`<form action="http://bidon.fr/traitement.php" method="post">`, le fichier PHP `traitement.php` du site `bidon.fr` est exécuté. Imaginons que ce fichier ressemble à cela ¹³ :

```
<?php
$dire = htmlspecialchars($_POST['dire']);
$a = htmlspecialchars($_POST['a']);
echo $dire, ' ', $a, ' !';
```

La variable globale `$_POST` donne accès aux données envoyées avec la méthode `POST` par leur nom. La méthode `echo` permet d'écrire un texte dans la page WEB qui sera retournée à l'utilisateur, via le protocole HTTP. Quand ce code PHP est exécuté, la sortie dans le navigateur de l'utilisateur sera « Bonjour Maman ! ».

Les navigateurs ne savent pas interpréter le code PHP. Si le formulaire est envoyé à une page PHP située dans l'ordinateur de l'utilisateur, le navigateur n'affichera pas le message attendu. Pour qu'il s'exécute, il est nécessaire de lancer l'exemple par l'intermédiaire d'un serveur PHP. On peut simuler un tel serveur PHP sur son ordinateur ; cette technique est utile pour tester les traitements en local avant de les installer sur un serveur distant. Pour tester mon fichier PHP, je l'ai installé sur le serveur qui héberge mon site Mathadomicile. Celui-ci m'a refusé l'envoi d'une requête avec l'attribut `POST`. J'ai donc modifié le fichier HTML pour qu'il transmette un attribut `GET` et aussi le fichier PHP pour qu'il utilise `$_GET` à la place de `$_POST` (les deux variables fonctionnent de façon similaires). J'ai ainsi obtenu l'affichage de ma salutation (l'adresse réelle de mon site s'est aussi affichée dans l'URL complétée). Du coup, comme le fichier PHP est resté sur mon site, vous pouvez l'utiliser également (il n'a pas d'autre intérêt que celui de renvoyer les données de ce très court formulaire).

```
<form method="get" action="http://mathadomicile.fr/formTraitement.php">
  <div>
    <label for="dire">Quelle salutation envoyer?</label>
    <input name="dire" id="dire" value="Hello">
  </div>
  <div>
    <label for="a">Qui voulez-vous saluer?</label>
    <input name="a" value="Moi-même">
  </div>
  <div>
    <input type="submit" value="Envoyer ma salutation">
  </div>
</form>
<style>
  form {width: 600px;}
  div {margin-bottom: 20px;}
  label {display: inline-block;
  width: 300px;
  text-align: right;
  padding-right: 10px;}
  button, input {float: right;}
</style>
```

Le fichier HTML et son apparence dans le navigateur

```
<?php
$dire = htmlspecialchars($_GET['dire']);
$a = htmlspecialchars($_GET['a']);
echo $dire, ' ', $a, ' !';
```

Le fichier PHP et le message qu'il affiche dans le navigateur

Le traitement minimaliste effectué dans cet exemple récupère les données, les formate et les retourne à l'utilisateur pour qu'elles s'affichent. Mais les traitements côté serveur vont bien au delà : on peut ranger les données dans une base de données, faire des statistiques, envoyer des mails, etc. On doit aussi effectuer des traitements de contrôle pour éviter les attaques malveillantes. Les formulaires HTML sont, en effet, l'un des principaux vecteurs d'attaque contre les serveurs. Celui-ci ne peut faire confiance à la validation côté client car il n'a aucun moyen de savoir ce qui se passe réellement du côté client. Il doit donc être très vigilant et mettre en œuvre des traitements spécifiques sur les données reçues.

Il y a de nombreux autres langages que le PHP pour gérer les formulaires côté serveur : Perl, Java, .Net, Ruby, etc. Il y a aussi des canevas spécialement conçus pour traiter les formulaires plus facilement : Symfony (pour PHP), Django (pour Python), etc. Ce domaine, très technique, est plutôt l'affaire des professionnels, aussi nous en resterons là sur ce sujet.

13. Pour des éléments de syntaxe PHP voir <https://www.php.net/manual/fr/language.variables.basics.php>