# Les fondamentaux Web

- Historique
- Organismes de normalisation
- Définition du HTML5
- Principe du HTML5
- Quand choisir HTML5/CSS3

# HISTORIQUE 1/2

- **1980** : Tim Berners-lee et Robert Caillau mettent au point un système de navigation Hypertext. (ancêtre des navigateurs)
- **1986** : Création du langage SGML (Standard General Markup Language) pour structurer des contenus divers, considéré comme le 1er langage à balise.
- **1991** : Tim Berners-lee met au point le protocole HTTP (Hyper Text Transfer Protocol), ainsi que le langage HTML (Hyper Text Markup Language). Naissance du World Wide Web (WWW).
- **1996** : 1ère spécification HTML 2.0 par le W3C (World Wide Web Concortium). le HTML 1.0 n'a jamais vraiment existé. Apparition du CSS (Cascade Style Sheet – feuille de style) créé par Hâkon Wium Lie.
- De **1997** à **1999** :
  - Spécification HTML 3.2, 4.0 et 4.01
  - Spécification CSS 2.0
  - Création du XML (eXtensible Markup Language)

# HISTORIQUE 2/2

- **2000** : Le W3C lance le XHTML 1.0, celui-ci possède exactement le même contenu que le html 4.01 la différence tient sur la syntaxe, le xhtml suivait les règles du xml. Exemple tous les attributs d'une balise ne s'exprime plus qu'en minuscule. (cette sortie coïncida avec l'essor des navigateurs compatible avec CSS).
- **2001** : Le W3C recommande le XHTML 1.1, celui-ci est entièrement en XML, cependant gros problème le navigateur le plus populaire du moment de peut pas l'interpréter (ie explorer). Le W3C perd pied avec la réalité des publications web. Ceci ne l'empêche pas de se lancer dans le XHTML 2.0 cependant celui-ci possède plusieurs problèmes techniques et ne répond toujours pas aux besoins développeurs du moment.
- **2004** : Scission au sein duW3C. Les représentants d'Opéra, Apple et Mozilla ne sont pas en accord avec les priorités du W3C. Proposition de reprise du développement HTML pour création d'application web mais celle-ci est rejetée. Les représentants d'Opéra (dont Ian Hickson) et autres forment leur propre groupe WHATWG (Web Hypertext Application Technilogy Working Group)
- **2006** : Tim Berners-Lee admet que la migration du HTML vers XML était une erreur. Quelques moisplus tard le W3C planche sur la version HTML 5 (avec espace) et continue cependant sur le XHTML 2.0. De son côté WHATWG travaille sur le HTML5 (sans espace).
- **2007** : Le W3C accepte les propositions de recommandations du WHATWG sur le HTML5 (sans espace).
- **2009** : Le format XHTML 2.0 est définitivement mort
- **2012** : La spécification HTML5 doit devenir « recommandation candidate », c'est-à-dire fin prête dans le discours normatif.

# Organismes de normalisation

- **IETF** : L'Internet Engineering Task Force, abrégée IETF, littéralement traduit de l'anglais en « Détachement d'ingénierie d'Internet » est un groupe informel, international, ouvert à tout individu, qui participe à l'élaboration de standards Internet. L'IETF produit la plupart des nouveaux standards d'Internet. L'IETF est un groupe informel, sans statut, sans membre, sans adhésion. Le travail technique est accompli dans une centaine de groupes de travail. En fait, un groupe est généralement constitué d'une liste de courrier électronique. L'IETF tient trois réunions par année.
- **W3C** : Le World Wide Web Consortium, abrégé par le sigle W3C, est un organisme de normalisation à but non-lucratif, fondé en octobre 1994 chargé de promouvoir la compatibilité des technologies du World Wide Web telles que HTML, XHTML, XML, RDF, SPARQL, CSS, PNG, SVG et SOAP. Fonctionnant comme un consortium international, il regroupe en 2012, 378 entreprises partenaires.
  Le W3C a été fondé par Tim Berners-Lee après qu'il eut quitté le CERN en octobre 1994. Le W3C a été fondé au MIT/LCS (Massachusetts Institute of Technology / Laboratory for Computer Science) avec le soutien de l'organisme de défense américain DARPA et de la Commission européenne.
- **WHATWG** : Le Web Hypertext Application Technology Working Group (ou WHATWG) est une collaboration non officielle des différents développeurs de navigateurs web ayant pour but le développement de nouvelles technologies destinées à faciliter l'écriture et le déploiement d'applications à travers le Web. La liste de diffusion du groupe de travail est publique et ouverte à tous. La Mozilla Foundation, Opera Software et Apple, Inc. en sont les premiers contributeurs. Ce groupe de travail se limite aux technologies qu'il estime imprésentables dans les navigateurs Web sur la base des implémentations actuelles, et particulièrement de celles d'Internet Explorer. Il se présente notamment comme une réponse à la lenteur supposée du développement des standards par le W3C et au caractère supposé trop fermé de son processus interne d'élaboration de spécification. Cependant, de nombreux participants à ce projet sont également des membres actifs du W3C, et le nouveau groupe de travail HTML du W3C a adopté en 2007 les propositions du WHATWG comme base de travail d'un futur HTML5.

# Définition du HTML5

- « **HTML5** (HyperText Markup Language 5) est la dernière révision majeure du HTML (format de données conçu pour représenter les pages web). Cette version a été finalisée le **28 octobre 2014**. HTML5 spécifie deux syntaxes d'un modèle abstrait défini en termes de DOM : HTML5 et XHTML5. Le langage comprend également une couche application avec de nombreuses API, ainsi qu'un algorithme afin de pouvoir traiter les documents à la syntaxe non conforme. Le travail a été repris par le W3C en mars 2007 après avoir été lancé par le WHATWG. Les deux organisations travaillent en parallèle sur le même document afin de maintenir une version unique de la technologie. Le W3C clôt les ajouts de fonctionnalités le 22 mai 2011, annonçant une **finalisation de la spécification en 2014**, et encourage les développeurs Web à utiliser HTML 5 dès ce moment. » (source wikipédia)
- Dans le langage courant, **HTML5** est une combinaison de nouvelles balises HTML, de propriétés CSS3, de JavaScript et de plusieurs technologies associées mais structurellement séparées de la spécification HTML5.

# Principe du HTML5

- Pour éviter les erreurs du passé le WHATWG à rédigé une liste de principes :
  - HTML5 devra supporter le contenu déjà existant (pas d'an zéro du HTML5)
  - S'il existe une façon répandue d'accomplir une tâche chez les webdesigners (même mauvaise), celle-ci doit être codifiée en HTML5 (« si ce n'est pas cassé, on répare pas »).
  - En cas de conflit on privilégie d'abord l'utilisateur, les webdesigners, les programmeurs (dans les navigateurs), les spécificateurs et enfin la beauté du code.

# Le balisage

- Structure d’une page (3 slides)
- Contenus & Nouvelles balises (7 slides)
- Exemple  « model_page.html » (6 slides)
- Exemple: architecture d'un site (15 slides)

# Structure d'une page 1/3

- **Les éléments de base**

  Le langage **HTML5** est une amélioration du langage HTML4, avec des simplifications par rapport à la version XHTML. Tout document peut donc débuter par la déclaration du doctype puis est suivi de l'élément racine html qui inclut les éléments head et body.

- Le « Doctype »

  La déclaration du doctype est traditionnellement utilisée pour spécifier le type de balisage du document. Celui de XHTML 1.0 était le suivant :

```
<!DOCTYPE html PUBLIC «-//W3C//DTD XHTML 1.0 Transitional//EN»
«http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd»>
```

Celui de HTML 4.01 était :

```
<!DOCTYPE html PUBLIC «-//W3C//DTD HTML 4.01/EN»
«http://www.w3.org/TR/html4/strict.dtd» >
```

Un des principes d'HTML5 étant la rétrocompatibilité avec les anciennes versions, sa définition se résume à un simple :

```
<!DOCTYPE html>
```

# Structure d'une page 2/3

- L'élément « **html** »

  Il est **l'élément racine** du document. Il est le parent de tous les autres, soit <head> et <body>. Celui–ci possède des attributs dont le plus utiles est lang. Celui-ci indique la langue utilisée par défaut dans la page. Cette valeur sera reconnue par les moteurs de recherche pour leur permettre d'indexer les pages du site en effectuant un tri par langue.

```
<html lang="fr">
```

- L'élément « **head** »

  Cette section donne quelques informations générales sur la page, comme son titre et l'encodage de la page (pour la gestion des caractères spéciaux). Ces informations sont les 2 seules obligatoires dans cette section. Si la balise <title> n'a pas changé, la déclaration de l'encodage a elle été simplifiée. Si avant nous avions :

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

  Maintenant en HTML5 :

```
<meta charset="utf-8" />
```

# Structure d’une page 3/3

- **Page minimale en HTML5**

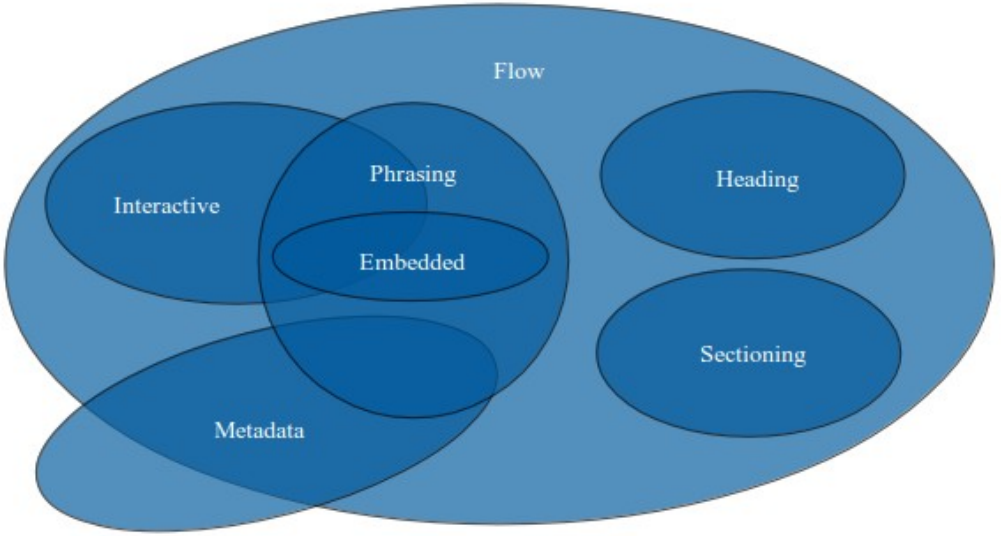
```
<!DOCTYPE html>

<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>page simple HTML5</title>
        <!--ici des ressources pour html5 : CSS/JavaScript/Keyword -->
    </head>
    <body>
        <!--ici contenu de ma page html5 -->
    </body>
</html>
```

# Contenus 1/7

- **Les contenus**

  Dans les versions précédentes de HTML les contenus étaient divisés en 2 grandes catégories, « en lignes » ou en « bloc », celui-ci est remplacé par un nouveau schéma ; les éléments HTML sont regroupés selon les catégories suivantes :

- Contenu Metadonnées (Metadata)
- Contenu de flux (Flow)
- Contenu de section (Sectioning)
- Contenu d'en-tête (Heading)
- Contenu de phrasé (Phrasing)
- Contenu embarqué (Embedded)
- Contenu interactif (Interactive)




*NB : Il est à noter que certains éléments peuvent apparaître dans plusieurs catégories. (intersections)*

# Les Contenus 2/7

- Contenu de métadonnées

Les métadonnées déterminent la présentation ou le comportement du contenu de la page. Elles servent aussi à configurer les relations entre ce document et d'autres (exemple : balise <meta> contenant des mots clé ou bien un descriptif de la page).

Les modifications apportées par HTML5 concernent les balises <link> et <script>. Concernant la balise script il n'est plus nécessaire d'indiquer le type. Celui-ci sera par principe considéré comme étant du javascript.

```
<script src="file/js"></script>
```

Il en va de même pour l'appel des feuilles de styles, l'attribut réellement nécessaire étant le média sur lequel il s'applique :

Les modifications apportées par HTML5 concernent les balises <link> et <script>. Concernant la balise script il n'est plus nécessaire d'indiquer le type. Celui-ci sera par principe considéré comme étant du javascript.

```
<link rel="stylesheet" href="styles.css" media="screen" />
```

Les différents média : all, print, screen, tv, aural, braille, enbossed, handled, projection, tty

# Les Contenus 3/7

- Contenu de flux

  Le contenu de flux représente les éléments qui sont considérés comme formant l'ensemble du contenu d'une page web. Un flux est en règle générale un texte ou un fichier embarqué comme une image ou bien une vidéo.

  Cette catégorie possède plusieurs nouveaux éléments :

  - **header** : spécifie une introduction, ou un groupe d'éléments de navigation pour le document.
  - **footer** : définit le pied de page d'un article ou un document. Contient généralement le nom de l'auteur, la date à laquelle le document a été écrit et / ou ses coordonnées.
  - **figure** : définit des images, des diagrammes, des photos, du code, etc.
  - **figcaption** : légende pour la balise <figure>.

```
<figure >
    <img src="assets/images/hitchcock.jpg" alt="Alfred Hitchcock" />
    <figcaption>Photo promotion du film <strong>Les oiseaux</strong></figcaption>
</figure>
```

# Les Contenus 4/7

- **canvas** : utilisé pour afficher des éléments graphiques, il faut utiliser un script pour l'animer.

- Contenu de section

  Ceci est une nouvelle catégorie. Le W3C décrit cette catégorie comme « définissant la portée des en-têtes et des pieds de pages ». Un contenu de section est un sous-ensemble d'un contenu de flux. Elle contient 4 nouveaux élements HTML5 :

  - **section** : définit les sections dans un document. Tels que les chapitres, en-têtes, pieds de page, ou toutes autres sections du document.
  - **article** : partie indépendante du site, comme un commentaire.
  - **aside** : associé à la balise qui le précède. Il pourrait être utilisé par exemple pour faire référence à des sujets complémentaires ou pour définir un terme spécifique.
  - **nav** : définit une section dans la navigation. Il contient les liens utiles à la navigation

# Les Contenus 5/7

- Contenu d'en-tête

  Contient tous les éléments d'en-tête standard comme <h1>, <h2> et ainsi de suite. En plus **HTML5** comprend l'élément :

  - **hgroup** : permet de définir des groupes de titres. Celui-ci ne peut contenir que les éléments de <h1> à <h6>. Il sert essentiellement à définir un plan, une table des matières.

    Entre les balises <body> insérer le code suivant:

```
<hgroup>
    <h1>La vie des castors</h1>
    <h2>...ou la passionnante aventure d'une espèce en danger</h2>
</hgroup>
```

- Contenu de phrasé

  Ce contenu est le texte du document, y compris les mises en évidence de passage à l'intérieur d'un paragraphe. Nouvelles balise HTML5 :

  - **mark** : définit un texte marqué.
    Entre les balises <body> insérer le code suivant :

```
<p>L'avantage d'avoir de <strong>bonnes</strong>chaussures de marche de
   randonné devient <em>extrèment</em> clair <mark>après 3 jours de marche</mark>.
</p>
```

# Les Contenus 6/7

- Contenu embarqué

  Un contenu embarqué importe une autre source dans la page, comme une image, une musique ou bien une vidéo. Ce contenu contient de nouvelles balises HTML5 permettant une alternative à FLASH :

  - **audio** : pour définir un son, comme la musique ou les autres flux audio (streaming).
  - **video** : Insérer un contenu vidéo en streaming.
  - **track** : Insérer un sous-titre (au format WebVTT) à une vidéo affichée avec la balise video .
  - **embed** : définit un contenu incorporé, comme un plug in.

- Contenu interactif

  L'une des balises les plus basiques <a>, est considérée comme un élément interactif ainsi que les éléments de formulaire tel <textarea> ou encore <button>. La balise <a> subit un changement. En effet dans les versions précédentes cette balise est inline (ancienne version ci-dessous) :

```
Bienvenue !
<h2><a href="http://www.kimo.fr">KIMO Industrie</a></h2>
<p>
    <a href="http://www.huiledecode.org">H.D.C : service en Logiciels Libres</a>
</p>
```

# Les Contenus 7/7

En HTML5, il est maintenant possible d'envelopper plusieurs éléments dans un seul élément <a> :

```
<a href="http://www.afci.fr">
    <h2>Huile de Code</h2>
    <p>formation en programmation informatique infographie vidéo,
       centre de formation informatique sur les langage de l'internet
       et du  web.
    </p>
</a>
```

- **Exemple: utilisation des nouveaux éléments :** Création page modèle

Nous allons dans cette partie créer une page modèle dont nous nous resservirons dans les prochains exercices. Celle-ci contiendra un simple en tête incluant une image ainsi qu'un titre et un sous-titre.

# Exemple « model_page.html » 1/6

- Fichier HTML et fichier CSS :

```html
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Modèle page HTML5</title>
        <link rel="stylesheet"
            href="assets/css/base.css"
            media="screen" />
    </head>
    <body>

    </body>
</html>
```

```css
@charset "utf-8";
/* CSS Document */
body {
    font: normal 90% "Trebuchet MS",
        Verdana,"Lucida Grande",
        Tahoma,Arial,Helvetica,
        Sans-Serif;
    color: #444;
    background:#f4f4f4;
    padding:0 0 2em 0;
    margin:0;
}
```

Nous avons attaché cette feuille de style à notre page html. Dans l'entête de notre page on inclu le lien suivant:

```html
<link rel="stylesheet" href="assets/css/base.css" media="screen" />
```

Enregistrer et visualiser le résultat grâce au protocole « file:/// ».

Nous allons réaliser le corps (<body>..</body>) de notre page modèle. Pour ce faire nous allons utiliser la balise « <header> (ainsi que <footer>) ». Elles ont été créées afin de donner au créateur et au développeur des options de balisage sémantique plus précis inhérent à la structure du document.

# Exemple « model_page.html » 2/6

Ces balises donnent une indication plus précise du contenu d'une section (auparavant celle-ci était contenu dans un  dont l'id permettait de connaître son contenu ex : ).

Dans le corps du document HTML :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Modèle page HTML5</title>
        <link rel="stylesheet"
            href="css/base.css"
            media="screen" />
    </head>
    <body>
        <header>
            <img src="images/logo_huiledecode.png" alt="logo huile de code" />
            <h1>HTML5</h1>
            <h2>Page modèle exercice</h2>
        </header>
    </body>
</html>
```

# Exemple « model_page.html » 3/6

- Dans la feuille de style :

```
header{
	color: #ee662a;
	background:white;
	border-bottom:2px dotted #ccc;
	margin:0;
	padding:0.8em;
	height: 220px;

}
header h1{
	font-size: 4em;
	font-weight: normal;
	margin:0;
}
header h2{
	font-size: 1.5em;
	font-weight: bold;
	margin: 0 0 2.5em 10em;
	color: #e74b25;
}
header img{
	float:left;
	margin-right:3em;
}
```

Si on visualise dans un navigateur récent aucun problème (chrome). Cependant si l'on essaie de le visualiser dans un navigateur antérieur (IE8) celui-ci ne reconnaît pas la balise header.

# Exemple « model_page.html » 4/6

- Voici le résultat de notre première page html intégrant une feuille de style CSS.


HTML5

Page modèle exercice

- Correctif Internet Explorer

  Pour pallier cette erreur nous allons procéder en 2 temps :

  *Création de la feuille de style associé* : css/correctif_IE.css

```
@charset "utf-8";
/* CSS Document */
header, section, aside, nav, footer, figure, figcaption {
	display: block;
}
```

# Exemple « model_page.html » 5/6

- Correctif Internet Explorer (suite)

  *création d'un document javascript* : js/correctif.js

```
// JavaScript Document
document.createElement('header');
document.createElement('footer');
document.createElement('nav');
document.createElement('aside');
document.createElement('section');
document.createElement('figure');
document.createElement('figcaption');
```

Enfin il ne nous reste plus qu'à inclure ces 2 nouveaux fichiers au sein de l'en-tête de notre document, après le lien de chargement de « base.css » :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Modèle page HTML5</title>
        <link rel="stylesheet" href="css/base.css" media="screen" />
        <link rel="stylesheet" href="css/correctif_IE.css" media="screen"/>
        <script src="js/correctif.js"></script>
    </head>
    (...)
</html>
```

# Exemple « model_page.html » 6/6

- Voyons si le correctif fonctionne, ci-dessous un screenshot avant d'intégrer les fichiers css et js dans le source, test avec Internet Explorer 8 :


- Ci-dessous un screenshot dans IE8 après avoir intégré les fichiers css et js :

# Exemple: architecture d'un site 1/15

- Nous allons à présent dans l'exemple qui suit créer une page HTML contenant l'ensemble des nouvelles balises :

  Création du fichier html: architectureSite.html à partir de la la page modele_page.html

  Création de la feuille de style : css/styles_architecture.css

- Puis modifier le titre de la page HTML («architecture site») et inclure la nouvelle feuille css :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Architecture site</title>
        <link rel="stylesheet" href="css/base.css" media="screen" />
        <link rel="stylesheet" href="css/correctif_IE.css" media="screen"/>
        <link rel="stylesheet" href="css/styles_architecture.css" media="screen" />
        <script src="js/correctif.js"></script>
    </head>
    (...)
</html>
```

# Exemple: architecture d'un site 2/15

- Le contenu de la page sera défini de la manière suivante :


- Nous allons voir avec un exemple concret la structure d'une page HTML/CSS avec les nouvelles balises intégré à HTML5.
- Choisissez un éditeur de code, je vois conseille « ATOM » il est libre et gratuit et écrit par les équipe de github, gage de qualité.

# Exemple: architecture d'un site 3/15

- **La balise** <nav> :


Nous allons à présent créer notre menu de navigation. Sous la balise </header> inclure le code suivant :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        (...)
    </head>
    <body>
        <header>
            (...)
        </header>
        <nav>
            <ul>
                <li><a href="#">ACCUEIL</a></li>
                <li><a href="#">A PROPOS</a></li>
                <li><a href="#">DIAPORAMA</a></li>
                <li><a href="#">CONTACT</a></li>
            </ul>
        </nav>
    </body>
</html>
```

# Exemple: architecture d'un site 4/15

- Définissons à présent la feuille de style associée : styles_architecture.css

```css
/* navigation */
nav {
	background-color:#f2662a;
	height:35px;
}
nav li {
	float:left;
	width:140px;
	height:35px;
	background-color:black;
	text-align: center;
	border-left: 1px white solid;
	border-right: 1px white solid;
	line-height:35px;
	list-style:none;
}
nav li a {
	color:white;
	text-decoration:none;
	display:block;
}
nav li a:hover{
	background-color:#f0462b;
	color:#ebebea;
}
```

← *Enregistrer et visualiser le résultat.*

[cf. slide suivante]

Il existe une alternative à la balise <nav> qui est la balise <menu>.

Cependant celle-ci avait été déclarée obsolète pour le XTHML. Celle-ci est censée permettre la création rapide de menu déroulant, cependant les navigateurs gèrent très peu cette balise.

Il est possible de réaliser le même effet avec liste à puces et feuilles de style.

# Exemple: architecture d'un site 5/15

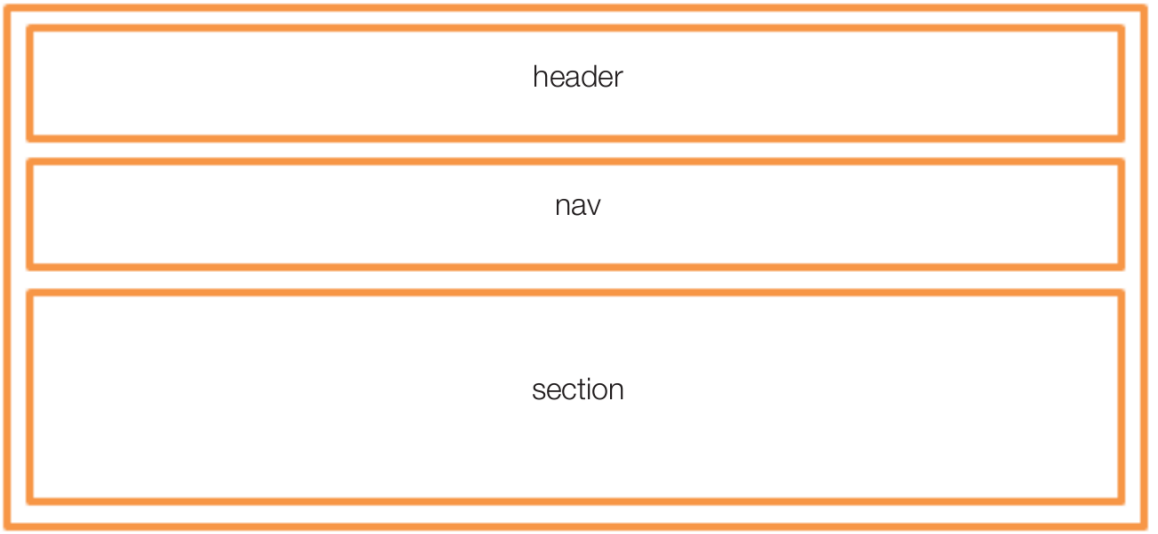
- Résultat <nav> dans Chromium :


- Résultat <nav> dans IE8 :

# Exemple: architecture d'un site 6/15

- La balise <section> :

Une section est définie comme étant un regroupement thématique de contenu. Dans notre exemple celui-ci sera notre conteneur principal. Ajoutons le contenu de notre section dans la page architecture (aidez-vous des textes lorem ipsum pour remplir celle-ci). A la suite de la balise </nav>




*Enregistrer et visualiser le résultat.*

```
<section>
      <h2>Mission du HTML5</h2>
      <p>Lorem ipsum ... fermentum.</p>
      <p>Integer accumsan ... purus.</p>
      <h2>What's new?</h2>
      <p>Par Stephane, posté le 21 Juin 2016</p>
      <img src="images/categorie.png"
            alt="Les contenus HTML5" />
      <p>Les différents conenus HTML5</p>
      <p>Donec ... ante.</p>
      <p>Fusce ... elit.</p>
      <p>Integer accumsan ... purus.</p>
</section>
```

```
/* section */
section {
      width:600px;
      float:left;
}
section p, h1, h2 {
      margin-left:20px;
      margin-right:20px;
}
section h2 {
      margin-top:15px;
}
```

# Exemple: architecture d'un site 7/15

Résultat <section> dans Chorme :


Résultat <section> dans IE8 :

# Exemple: architecture d'un site 8/15

- **La balise** <footer> :

Après la balise section nous ajoutons un pied de page sur l'ensemble du document.

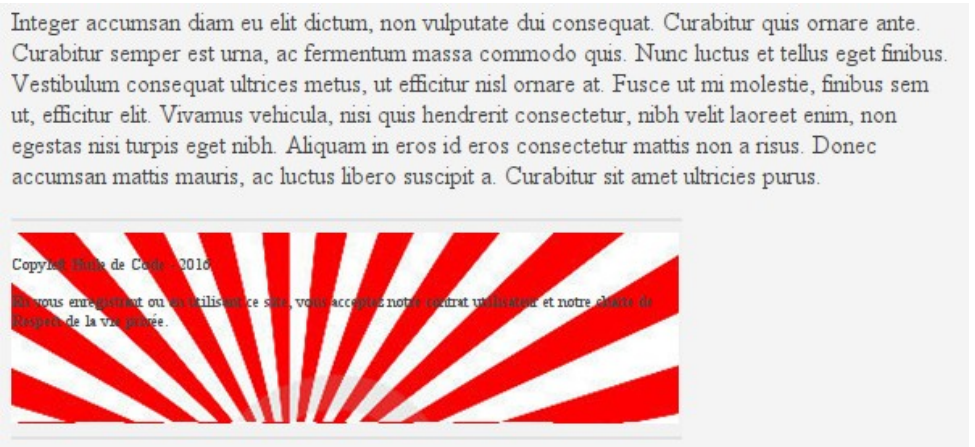
```
<!DOCTYPE html>
<html lang="fr">
      (...)
            <section>(...)</section>
            <footer>
                  <p>Copyleft Huile de Code - 2016</p>
                  <p>En vous enregistrant ou en utilisant ce site,
                    vous acceptez notre contrat utilisateur et notre
                    charte de Respect de la vie privée.</p>
            </footer>
      </body>
</html>
```

```
footer{
      clear: both;
      width: 400px;
      margin-left: 20px;
      font-size: 10px;
      background: #f2f2f2 url(file:../images/footer_background.jpg) no-repeat left;
      height: 118px;
      padding-top: 10px;
      border-top:#e2e2e2 solid 2px;
      border-bottom:#e2e2e2 solid 2px;
}
```

# Exemple: architecture d'un site 9/15

Résultat <footer> dans Chrome :


Résultat <footer> dans IE8 :




- **La balise** <article> :

  - **Le définition du W3C** : indique qu'il s'agit d'une composition autonome, indépendante du reste, dans un document, une application ou d'un site, et qu'elle est par principe distribuable ou réutilisable en soi. Il peut par exemple s'agir d'un post dans un forum, d'un article de presse, d'une entrée, d'un blog, d'un commentaire soumis par un utilisateur ou tout autre contenu autonome.

  - **Le concept clé** : est qu'un article est un contenu en soi. Celui-ci peut être facilement republié dans différents contextes. Il pourra être multi-diffusé dans divers formats, tel qu'un flux RSS, dans un email, via des applications mobiles.

# Exemple: architecture d'un site 10/15

- Repérer au milieu de votre <section> le 2ème <h2> et entourer ce contenu par la balise <article>.

```
<section>
      (...)
      <article>
            <h2>What's new ?</h2>
            <p>Par Stephane, posté le 21 Juin 2016</p>
            (...)
            <p>Integer accumsan diam ... ultricies purus</p>
      </article>
      (...)
</section>
```

- En HTML5 un article peut posséder son propre en-tête et pied de page

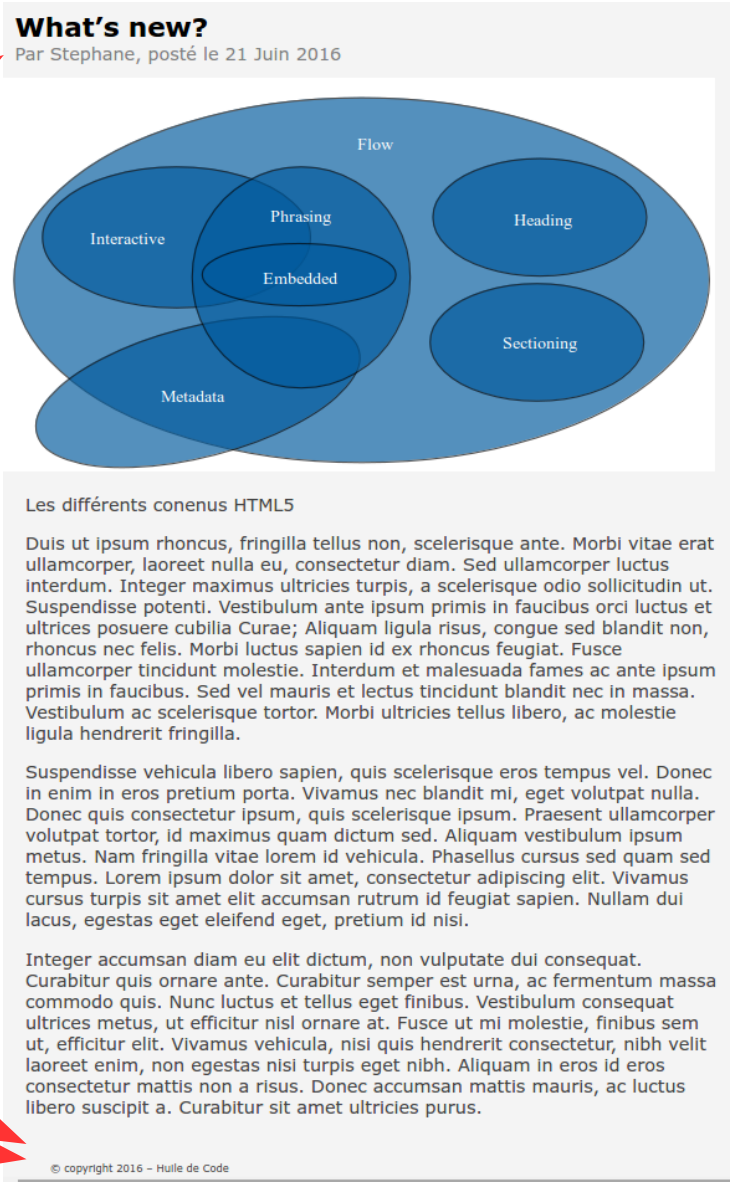
```
<article>
      <header>
            <h2>What's new ?</h2>
            <p>Par Stephane, posté le 21 Juin 2016</p>
      </header>
      (...)
      <footer>
            <p><small>© copyright 2016 – Huile de Code</small></p>
      </footer>
</article>
```

# Exemple: architecture d'un site 11/15

- Voyons le fichier CSS correspondant à cette structure

```css
/* article */
article header{
	background:none;
	height:auto;
	border-bottom:none;
}
article header h2{
	color:black;
	margin:0;
}
article header p{
	color: gray;
	margin: 0;
}
article footer{
	background:none;
	height:auto;
	border:none;
}
article footer p{
	font-size:10px;
}
```

- Résultat dans chromium :



# Exemple: architecture d'un site 12/15

- **La balise** <aside> :

  Cette balise peut être utilisée de 2 manières :

  - L'une correspond au placement classique d'une barre latérale dans une page web.
  - La seconde concerne une zone de contenu complémentaire, non indépendant, à l'intérieur d'une <section>.

1er Cas : *Enregistrer et visualiser le résultat.*


- Ajouter dans le document html « architectureSite.html » après a balise </nav>

```
      (...)
</nav>
<!-- bandeau gauche -->
<aside id="main_aside">
      <h3>Dernières recommandations</h3>
      <p>Duis ut ipsum ... fringilla. </p>
      <p> Nunc quam elit, ... aliquam. </p>
</aside>
      (...)
```

```
/* aside */
#main_aside{
      margin-top:15px;
      float: left;
      width: 300px;
      background-color: #378059;
      padding:7px;
      color:white;
}
```

# Exemple: architecture d'un site 13/15

- **2ème cas :**

Dans la section <article> juste avant le 2ème gros paragraphe ajouter le code suivant :
Ajouter ce code à « architecturSite.html »  Résultat avec les 2 cas : <aside>

```
(…)<aside id="article_aside">
      <h3>Plus d'infos ?</h3>
      <p>
            Lisez les recommandations du W3C.
            <a href="#">Lien.</a>
      </p>
</aside>(…)
```

```
#article_aside { /* aside */
      float: left;
      width: 12em;
      background: #F9A890;
      padding: 10px;
      margin: 0 20px;
}
#article_aside h3{
      color:#378059;
      margin:5px 0 5px 0;
}
#article_aside p{
      margin: 0;
}
```

# Exemple: architecture d'un site 14/15

- **La balise** <figure>

  Parmi les nouveaux éléments certains donnent davantage de sens à nos pages web. Par exemple <figure> et <figcaption> permettent d'identifier des images et des légendes associées à l'intérieur de notre contenu. Dans la partie <article> localiser la balise image et modifier le contenu pour obtenir :

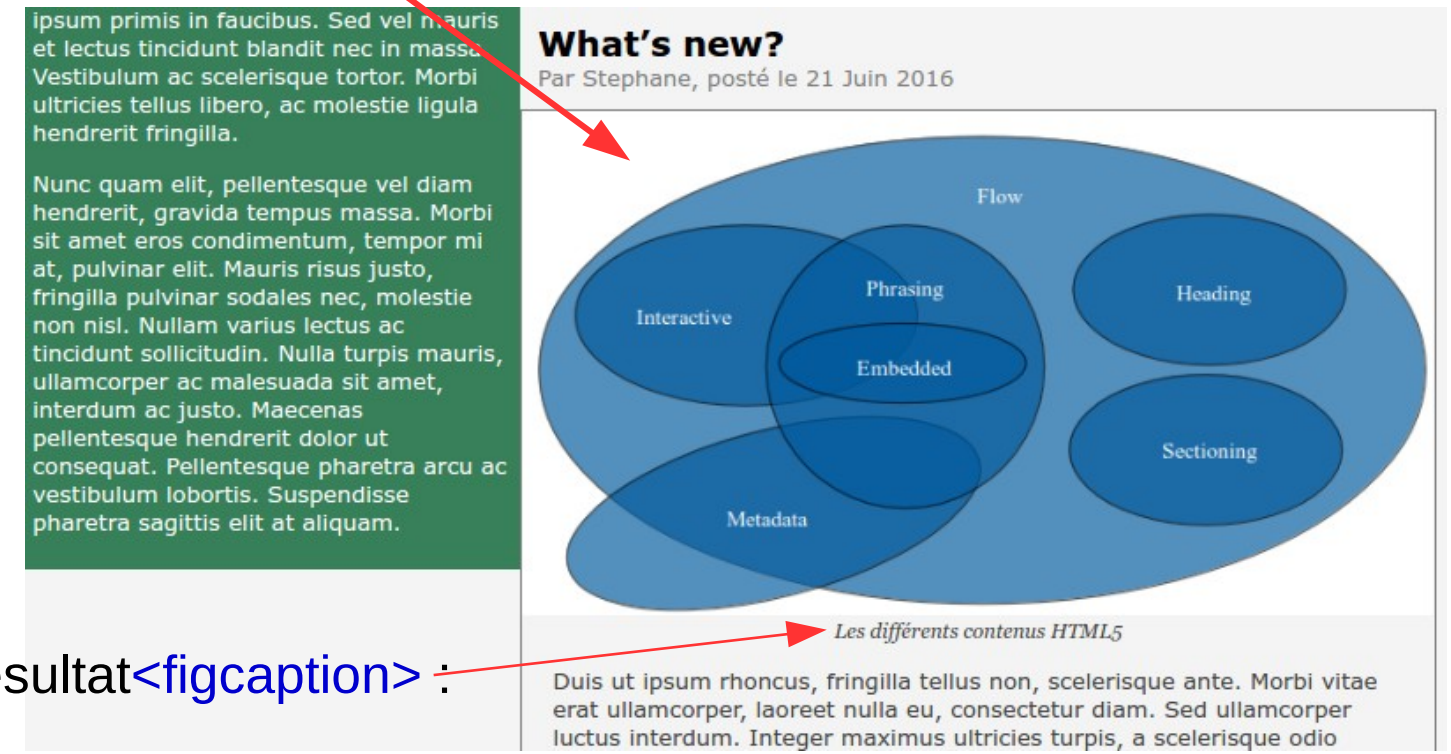
```
    (...)
<figure>
    <img src="assets/images/categorie.jpg" alt="Les contenus HTML5" />
    <figcaption>Les différents conenus HTML5</figcaption>
</figure>
    (...)
```

- Dans le CSS :

```
/*figure */
figure {
    float: right;
    border: 1px solid gray;
    padding: 0.25em;
    margin: 0 1.5em 1.5em 0;
}
figcaption {
    text-align:center;
    font: italic 0.9em Georgia, "Times New Roman", Times, serif;
}
```

# Exemple: architecture d'un site 15/15

- Résultat balise <figure> :




- Résultat<figcaption> :

- **La balise** <time> :

  Au début de <article> nous avons une date de publication. Le problème avec ce format est qu'il n'est pas dans un format lisible par une machine. HTML5 ajoute l'élément <time>. L'attribut **datetime** formaté AAAA-MM-DD. Celui-ci sera reconnu par un système informatique.il est également possible d'ajouter l'attribut pubdate.

```
<p>Par Stéphane, posté le <time datetime="2016-06-21" pubdate>21 Juin 2016</time>
```

# Les formulaires

- Composant d'un formulaire
- Place holder, required, pattern & validation
- Les nouveaux types <input>

Les formulaires comptent parmi les plus anciens et les plus familiers exemples d'interactivité sur le web.

L'élément FORM a été introduit dans le langage HTML en 1993 et les contrôles associés font partie de l'environnement quotidien de l'utilisateur.

Les nouveaux composants HTML5 ajoutent des fonctionnalités que les concepteurs web incorporaient traditionnellement par d'autres moyens, comme javascript ou flash. Il est courant dans un formulaire de rendre tel ou tel champ obligatoire, d'en vérifier le contenu avant de soumettre un formulaire.

Ces fonctionnalités sont maintenant intégrer à HTML5.

# Composant d'un formulaire 1/12

- Nous allons créer un simple formulaire permettant de recevoir un nom et un prénom :
- A partir du modèle de page créé précédemment, insérer le formulaire suivant

  *Création du fichier html*: formulaire1.html à partir de la page modele_page.html
- Insérer après la balise </header>

```
<div id="wrap">
    <form id="mon_formulaire" action="" method="post">
        <label for="nom">Nom : </label>
        <input type="text" name="nom" id="nom" /><br/>
        <label for="prenom">Prénom : </label>
        <input type="text" name="prenom" id="prenom" /><br/>
        <input type="submit" value="Valider" />
    </form>
</div>
```

# Composant d'un formulaire 2/12

- Voici le résultat, sobre pour le moment


HTML5

Page modèle exercice

Nom :
Prénom :
Valider

- La balise formulaire **<form>** contient 3 attributs :
  - **id** : permet d'associer une style CSS ou bien d'être identifier par JavaScript pour un traitement.
  - **action** : spécifie une url à laquelle le formulaire va poster les données du formulaire
  - **method** : (POST/GET) précise le mode d'nvoie des données, ici un POST.

# Composant d'un formulaire 3/10

- Nous avons ensuite dans le code du « formulaire1.html » deux éléments <input> de type **text**, ceux-ci représente les champs dans lesquels seront saisie par l'utilisateur les valeurs. Ils possèdent 2 attributs supplémentaires :
  - **id**
  - **name**

  Ces 2 attributs devront toujours être présents afin de faciliter le traitement des informations que cela soit en javascript ou dans un système de Template (génération dynamique du code HTML).
- Et enfin il reste un dernier élément <input> celui-ci étant de type **submit**. C'est lui qui donnera l'ordre d'envoi du formulaire au serveur lorsque l'on cliquera dessus. Le navigateur le formalise par un Bouton gris.

# Composant d'un formulaire 4/10

- Nous allons tout de même un peu customiser notre formulaire en ajoutant l'élément <fieldset> permettant de regrouper des éléments de formulaire et y ajouter une légende.

```
<div id="wrap">
    <form id="mon_formulaire" action="" method="post">
        <fieldset>
            <legend> Informations personnelles </legend>
            <label for="nom">Nom : </label>
            <input type="text" name="nom" id="nom" /><br/>
            <label for="prenom">Prénom : </label>
            <input type="text" name="prenom" id="prenom" /><br/>
            <input type="submit" value="Valider" />
        </fieldset>
    </form>
</div>
```


On remarque que <fieldset> permet d'aligner les zones de saisie et d'avoir un formulaire standardisé avec groupes.

# Composant d’un formulaire 5/10

- Style CSS pour notre formulaire

```
#wrap {
    padding: 0 3em 3em 3em;
    margin:auto;
}
#mon_formulaire {
    margin-top: 5px;
    padding: 10px;
    width: 400px;
}
#mon_formulaire label{
    float:left;
    font-size:13px;
    width:110px;
}
```

# Composant d'un formulaire 6/10

- Il est à noter que la balise <form> s'enrichie de 2 nouveaux attribut :
  - **autocomplete** : (on/off) permet de lancer la saisie semi-automatique
  - **novalidate** : (on/off) Si présent le formulaire n'est pas validé lorsqu'il est soumis

- *TP : Réaliser le formulaire de contact de notre petit site internet, il contiendra les champs suivant (25 mn) :*
  - *Nom,*
  - *Prénom,*
  - *Téléphone portable & Téléphone Fixe*
  - *Email*
  - *Zone de saisie multi-lignes en HTML. (<textarea>)*

# Composant d'un formulaire 7/10

- **HTML5** introduit de nombreuses nouveautés pour les formulaires pour améliorer l'aide à la saisie et les contrôles disponibles pour l'utilisateur. Plusieurs attributs simples à mettre en place améliorent la prise en charge des formulaires, tout en se passant de JavaScript.
  - Placeholder : est un attribut qui permet de renseigner un texte indicatif par défaut dans un champ de formulaire.
  - Modifier la page et ajouter l'attribut **placeholder** sur chaque champ de saisie.

```
<label for="nom">Nom : </label>
<input type="text" name="nom" id="nom" placeholder="votre nom" /><br/>
```

  - L'attribut **placeholder** peut être placé sur les éléments :
    - <input> : de type text, search, password, url, tel, email
    - <textarea>
  - Compatibilité : Firefox 4.0+, Opera 11.01+, Chrome 3.0+, Safari 3.0+, ie 10+

# Composant d'un formulaire 8/10

- Les champs **requis :**

  L'attribut required permet de rendre obligatoire le remplissage d'un champ et bloquer la validation du formulaire si l'un des champs (concernés par cet attribut) n'a pas été renseigné.

- Ajouter l'attribut **required** sur chaque champ de saisie

```
<div id="wrap">
    <form id="mon_formulaire" action="" method="post">
        <fieldset>
          <legend> Informations personnelles </legend>
          <label for="nom">Nom : </label>
          <input type="text" name="nom" id="nom" placeholder="Votre nom" required/>
          <br/>
          <label for="prenom">Prénom : </label>
          <input type="text" name="prenom" id="prenom"
                 laceholder="Votre nom" required/>
          <br/>
          <input type="submit" value="Valider" />
        </fieldset>
    </form>
</div>
```

# Composant d'un formulaire 9/10

- Appliquer un style au placeholder, nous allons pouvoir appliquer un CSS sur le champ de saisi afin d'indiquer que celui-ci est obligatoire

```
[required] {
	border: 1px solid orange;
}
[type="text"]::-webkit-validation-bubble-arrow {
	background-color: #f4f4f4;
	border: 1px solid #D8000C;
	border-width: 1px 0 0 1px;
	box-shadow: none;
}
[type="text"]::-webkit-validation-bubble-icon {
	display:none;
}
[type="text"]::-webkit-validation-bubble-message {
	background: #f4f4f4;
	border: 1px solid #D8000C;
	color: #D8000C;
	font-size: 100%;
	box-shadow: 0 1px 5px #bbb;
	text-shadow: 0 1px 0 #fff;
}
```

# Composant d'un formulaire 10/10

- Les **patterns** et la validation à la volée : Les champs requis non complétés ou ne respectant pas une certaine règle de syntaxe définie par le type du formulaire, ou par l'attribut pattern bloquent le processus d'envoi des données d'un formulaire.

```
<label for="telephone">Téléphone : </label>
<input type="tel" name="telephone"
       pattern="^((\+\d{1,3}(-| )?\(?\d\)?(-| )
       ?\d{1,5})|(\(?\d{2,6}\)?))(-| )?(\d{3,4})(-| )?(\d{4})
       (( x| ext)\d{1,5}){0,1}$"/>
```

- Pour connaître les « Patterns » en fonction du type de champs, je vous conseille le site :

http://html5pattern.com

# JavaScript - Introduction

- Le langage Javascript a été mis au point par Netscape (Brendan EICH) en Décembre 1995. Microsoft ayant sorti son propre langage quelques temps après (JScript), un standard a été créé, ECMAScript et le JavaScript des navigateur en est une « implémentation » . Actuellement, la version du langage est le JavaScript 1.6.
- C'est un langage interprété. Même si depuis peu il existe des JIT compiler notamment dans Node.js et V8 le moteur de Google.

```
<!DOCTYPE html>
<head>
  <script>
    document.write('Hello World !');
  </script>
</head>
<body>

</body>
```

```
<!DOCTYPE html>
<head> (...)</head>
<body>
  <script>
    document.write('Hello World !');
  </script>
</body>
<script src="js/helloworld.js"></script>
```

# JavaScript - Introduction

- helloworldJS.html

```
<!DOCTYPE html>
<head>
  <script src="js/helloworld.js"></script>
</head>
<body>

</body>
```

helloworld.js

```
document.write('Hello World !');
```

- On utilisera pas JavaScript de cette façon, on va utiliser des fonctions et les appeler, modifions notre « helloworld.js » pour encapsuler l'écriture dans l'objet document :

```
<!DOCTYPE html>
<head>
  <script src="js/helloworld.js"></script>
</head>
<body onload="hello();">

</body>
```

```
function hello() {
    document.write('Hello World !');
}
```

# JavaScript - Introduction

- Voyons comment déclencher un Javascript sur un autre évènement : onchange

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <script src="js/helloworld.js"></script>
</head>
<body onload="">
  <select name="liste_pays" onchange="whoisselected(this)">
    <option value="FR">Fance</option>
    <option value="ES">Espagne</option>
    <option value="AL">Allemagne</option>
    <option value="BE">Belgique</option>
    <option value="CH">Suisse</option>
  </select>
</body>
```

- La fonction « whoisselected() »

```
function hello() {
    document.write('Hello World !');
}

function whoisselected(liste) {
    alert('le pays sélectionné est : ' + liste.options[liste.selectedIndex].value);
}
```

# JavaScript - Introduction

- **Porté d'une variable**
  - Les variables déclarées tout au début du script, en dehors et avant toutes fonctions, seront toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle. On pourra donc les exploiter partout dans le script.
  - Dans une fonction, une variable déclarée par le mot clé var aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le script. D'où son nom de locale. Par contre, toujours dans une fonction, si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée sera globale.

# JavaScript - Introduction

- Les opérateurs de calcul

Pour **x = 11**

| Signe | Nom | Signification | Exemple | Résultat |
|---|---|---|---|---|
| + | plus | addition | x + 3 | 14 |
| - | moins | soustraction | x - 3 | 8 |
| * | multiplié par | multiplication | x * 2 | 22 |
| / | divisé par | division | x / 2 | 5.5 |
| % | modulo | reste de la division par | x % 5 | 1 |
| = | affectation | a la valeur | x = 5 | 5 |

# JavaScript - Introduction

- Les principaux types JavaScript sont :
  - Les nombres : Number
  - Les chaînes de caractères : String
  - Les booléens : Boolean
  - Les fonctions : Function
  - Les objets : Object
  - Les Tableaux : Array

Ce sont des Objets, ils possèdent des méthodes qui permettent de réaliser des traitements, exemple sur String :

```
return 'chat'.charAt(2); // renvoie "a"
return 'chat'[2]; // renvoie "a"
var monthString = "Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec";
var tabMonth = monthString.split(','); //convertir monthString en tableau de mois
console.log('tabMonth : ', tabMonth) ;
```

# JavaScript - Introduction

- Les opérateurs de comparaison

| Signe | Nom | Exemple | Résultat |
|---|---|---|---|
| == | Egal | x == 11 | true |
| < | Inférieur | x < 11 | false |
| <= | Inférieur ou égal | x <= 11 | true |
| > | Supérieur | x > 11 | false |
| >= | Supérieur ou égal | x >= 11 | true |
| != | Différent | x != 11 | false |

# JavaScript - Introduction

- Les opérateurs associatifs

| Signe | Description | Exemple | Signification | Résultat |
|---|---|---|---|---|
| += | plus égal | x += y | x = x + y | 16 |
| -= | moins égal | x -= y | x = x - y | 6 |
| *= | multiplié égal | x *= y | x = x * y | 55 |
| /= | divisé égal | x /= y | x = x / y | 2.2 |

- Les opérateurs logiques

| Signe | Nom | Exemple | Signification |
|---|---|---|---|
| && | et | (condition1) && (condition2) | condition1 et condition2 |
| \|\| | ou | (condition1) \|\| (condition2) | condition1 ou condition2 |

- Les opérateurs d'incrément

| Signe | Description | Exemple | Signification | Résultat |
|---|---|---|---|---|
| x++ | incrémentation | y = x++ | y = x + 1 | 6 |
| x-- | décrémentation | y= x-- | y = x - 1 | 4 |

# JavaScript - Introduction

- Les Fonctions
  - On peut les déclarer de plusieurs façons pour le moment c'est la déclaration « classique » qui nous intéresse (comme on la vue au début de ce chapitre)

```
function <nom_de_la_fonction>(param1, param2) {
    /* corps de la fonction liste d'instructions */
}
```

- Appeler une fonction
  - Il faut que la fonction ai été déclarée avant l'appel, car le code est lu de haut en bas

```
<nom_de_la_fonction>('toto', 'titi') ;
```

# JavaScript - Introduction

- Les structures de contrôle
  - Structure séquentielle

| Algorithme | Code JS |
|---|---|
| DEBUT<br>Instruction1<br>Instruction2<br>FIN | {<br>Instruction1<br>Instruction2<br>} |

- Les instructions conditionnelles
  - Le « if … else …. »

| Algorithme | Code JS |
|---|---|
| SI condition<br>ALORS Instructions1<br>SINON Instructions2<br>FINSI | If (condition) {<br>Instruction1<br>}<br>else {<br>Instruction2<br>} |

# JavaScript - Introduction

- L'express réduite «  (condition) ? Alors : Sinon »

| Algorithme | Code JS |
|---|---|
| SI condition<br>ALORS Instructions1<br>SINON Instructions2<br>FINSI | (condition) ?<br>instruction 1 :<br>instruction 2 |

- L'itération for

| Algorithme | Code JS |
|---|---|
| POUR i = valeur<br>initiale A i = valeur<br>finale<br>REPETER instructions<br>FIN POUR | For (val initiale ;<br>condition ;<br>incrémentation) {<br>Instructions<br>} |

# JavaScript - Introduction

- L'instruction switch

```
switch (expression) {
  case valeur1:
    // Instructions à exécuter lorsque le résultat
    // de l'expression correspond à valeur1
    instructions1;
    [break;]
  case valeur2:
    // Instructions à exécuter lorsque le résultat
    // de l'expression correspond à valeur2
    instructions 2;
    [break;]
  ...
  case valeurN:
    // Instructions à exécuter lorsque le résultat
    // de l'expression à valeurN
    instructionsN;
    [break;]
  default:
    // Instructions à exécuter lorsqu'aucune des valeurs
    // ne correspond
    instructions_def;
    [break;]
}
```

Une instruction switch commence par évaluer l'expression fournie (cette évaluation ne se produit qu'une fois). Si une correspondance est trouvée, le programme exécutera les instructions associées.

L'instruction break peut optionnellement être utilisée pour chaque cas et permet de s'assurer que seules les instructions associées à ce cas seront exécutées. Si break n'est pas utilisé, le programme continuera son exécution avec les instructions suivantes (des autres cas de l'instruction switch).

# JavaScript - Introduction

- Fonction de boucle de temps : setIntervale.html

```
<!DOCTYPE html>
<html>
<head>
<title>setInterval/clearInterval example</title>
<script>
var intervalID;

function changeCouleur() {
  intervalID = setInterval(flashText, 1000);
}

function flashText() {
  var elem = document.getElementById("ma_boite");

  if (elem.style.color == 'red') {
    elem.style.color = 'blue';
  } else {
    elem.style.color = 'red';
  }
}

function stopTextColor() {
  clearInterval(intervalID);
}
</script>
</head>
```

# JavaScript - Introduction

- La suite du code du fichier de test setInterval.html

```
<body onload="changeCouleur();">
    <div id="ma_boite">
        <p>Salut tout le monde</p>
    </div>
    <button onclick="stopTextColor();">Stop</button>
</body>
</html>
```

- Récupérer des élements du DOM

```
(…)
<script>
function loadElements() {
        var listeElements = document.getElementsByTagName('a');
        for (var i=0;i<listeElements.length;i++) {
            window.open(listeElements[i]['href'], listeElements[i]['innerText']);
        }
}
</script>
</head>
<body onload="loadElements()">
  <a href="http://www.huiledecode.org">lien1</a>
  <a href="http://www.google.fr">lien2</a>
  <a href="http://www.caniuse.com">lien3</a>
  <a href="http://www.kimo.fr">lien4</a>
</body></html>
```

# JavaScript - Introduction

- *TP : En utilisant ce que l'on vient de voir sur JavaScript réaliser le diaporama de notre mini site architectureSite.html*
- *On pourra jouer sur la propriété « opacity » du « style » d'une image (img) (img.style.opacity) et utiliser la propriété « transition : opacity .5s ; » (30 mn)*
- *Un exemple de css pour les images qui fonctionne*

```
img {
  position: absolute;
  transition: opacity .5s;
  opacity: 0;
}
```

# JavaScript - Introduction

- Une correction au TP : le fichier diaporama.html

```
<!DOCTYPE html>
<html>
<head>
  <title>setInterval/clearInterval example</title>
  <link rel="stylesheet" href="css/diaporama.css" media="screen" />
  <script src="js/diaporama.js"></script>
</head>
<body onload="slideShow()">
   <img src="http://www.fillmurray.com/400/300" alt="Fill Murray">
   <img src="http://www.placecage.com/400/300" alt="Place Cage">
</body>
</html>
```

- Le fichier JavaScript : diaporama.js

```
function slideShow() {
  var current = 0,
      slides = document.getElementsByTagName("img");

  setInterval(function() {
    for (var i = 0; i < slides.length; i++) {
      slides[i].style.opacity = 0;
    }
    current = (current != slides.length - 1) ? current + 1 : 0;
    slides[current].style.opacity = 1;
  }, 3000);
}
```

# JavaScript - Introduction

- XMLHttpRequest

  C'est un objet JavaScript qui permet de faire un appel au serveur pour récupérer des données sans avoir à recharger la page.

  - Exemple de base :

```
<!DOCTYPE html>
<html>
<head>
  <title>setInterval/clearInterval example</title>
  <link rel="stylesheet" href="css/diaporama.css" media="screen" />
  <script src="js/ajax.js"></script>
</head>
<body onload="getDataSynchrone()">

</body>
</html>
```

- Le fichier JavaScript « ajax.js » qui contiendra la fonction **`getDataSynchrone()`**

# JavaScript - Introduction

- Le fichier « ajax.js »

```
function getDataSynchrone () {
  var req = new XMLHttpRequest();
  req.open('GET', 'http://localhost:8080/data.json', false);
  req.send(null);
  if(req.status == 200) console.log('data reçue :' + req.responseText);
}

function getDataAsynchrone() {
  var req = new XMLHttpRequest();
  req.open('GET', 'http://localhost:8080/data.json', true);
  req.onreadystatechange = function (evt) {
  if (req.readyState == 4) {
     if(req.status == 200)
      console.log('retour data: ' + req.responseText);
     else
      console.log('Erreur pendant le chargement de la page.\n');
  }
 for (var i=0;i<1000;i++) {
   //console.log('valeur de i: ' + i);
 }
};
req.send(null);
}
```

# JavaScript - Introduction

- Fichier de données JSON : « data.json »

  Le format JSON est un format de notation Objet basé sur JavaScript

  - Voici un exemple de fichier JSON qui va nous retourner des données d'une personne

```
{
  "personne" : {
      "nom": "MASCARON",
      "prenom": "Stéphane",
      "fonction": "Architecte Logiciel",
      "date_naissance" : "22/04/2970"
  }
}
```

- *TP-1 : En utilisant ce principe, créez un formulaire permettant d'affichier les données du JSON ci-dessus. Et remplissez le formulaire*
- *TP2 : A l'aide du fichier countries.json que je vous ai passé, vous aller créer une liste déroulante qui contient tous les pays*

*/*** A FAIRE : Correction des exercices ***/*

# JavaScript - Introduction

- La gestion des cookies en JavaScript
  - Le cookie, cette être étrange qui trace tout vos faits et gestes sur Internet. Voyons comment l'exploiter avec JavaScript.
  - Les cookies ont été inventés par Netscape afin de donner une "mémoire" aux serveurs et navigateurs Web. Le protocole HTTP, qui gère le transfert des pages Web vers le navigateur ainsi que les demandes de pages du navigateur vers le serveur, est dit state-less (sans état).
  - Il existe un objet « cookie », sous-objet de document, qui contient la liste des cookies enregistrés pour le domaine visité. Cette liste de cookies est donnée sous la forme d'une chaîne de caractères.

# JavaScript - Introduction

- Voyons deux fonctions permettant respectivement d'écrire et lire un cookie.
  - Ecriture du cookie (appel dans onload pour tester)

```
function setCookie(sName, sValue) {
  var today = new Date(), expires = new Date();
  expires.setTime(today.getTime() + (365*24*60*60*1000)); // date du jour + 1 ans
  document.cookie = sName + "=" + encodeURIComponent(sValue) +
                    "; expires=" + expires.toGMTString()+  "; path=/";
}
```

- Code html pour tester l'écriture  et la lecture d'un cookie

```
<!DOCTYPE html>
<html>
<head>
  <title>cookies exemple</title>
  <script src="js/cookies.js"></script>
</head>
<body onload="setCookie('kimo', 'formation')">
  <button onclick="getCookies()">Cookies</button>
</body>
</html>
```

- Pour la lecture juste un : « document.cookie »

# JavaScript - Introduction

- Stockage des données locales : WebStorage
  - Nous connaissions les cookies. Maintenant nous disposons de Web Storage (ou DOM Storage) pour stocker des données locales.
  - C'est une technique plus puissante que les cookies, qui sont limités en taille, quelques Ko contre plusieurs Mo pour Web Storage.
  - Web Storage met à disposition deux interfaces nommées sessionStorage et localStorage. Ces dernières ne sont plus véhiculées sur le réseau HTTP et elles sont facilement accessibles (lecture, modifications et suppression) en JavaScript.

# JavaScript - Introduction

- Stockage de session **sessionStorage**
  - L'interface sessionStorage mémorise les données sur la durée d'une session de navigation, et sa portée est limitée à la fenêtre ou l'onglet actif. Lors de sa fermeture, les données sont effacées. Contrairement au cookies, il n'y a pas d'interférence. Chaque stockage de session est limité à un domaine.
- Stockage local localStorage
  - L'interface localStorage mémorise les données sans limite de durée de vie. Contrairement à sessionStorage, elles ne sont pas effacées lors de la fermeture d'un onglet ou du navigateur. La portée de localStorage est de facto plus large : il est possible de l'exploiter à travers plusieurs onglets ouverts pour le même domaine ou plusieurs fenêtres (du même navigateur).

- Les avantages du WebStorage :
  - Stocker rapidement des données en cache sans faire intervenir le serveur (par exemple via AJAX),
  - augmenter la performance ressentie et faciliter les développements
  - se passer des cookies et du trafic HTTP supplémentaire qu'ils représentent (un cookie est envoyé à chaque requête effectuée sur un domaine),
  - exploiter un espace alloué plus important que la limite imposée par les cookies (fixée à 4 Ko),
  - retrouver des données immédiatement à la reconnexion ou les mémoriser pour éviter la perte s'il y a déconnexion.

# JavaScript - Introduction

- Avertissements :
  - Attention : les données ne sont pas cryptées, accessibles facilement à tout utilisateur ayant accès au navigateur, et modifiables de la même façon. Il ne faut donc pas y placer d'informations sensibles.
  - Certains navigateurs exigent de consulter la page appelant le stockage via un domaine, (c'est-à-dire avec une url en http://, que ce soit localhost ou bien un nom de domaine complet) et non pas en fichier local (adresse file://). Sinon, une exception de sécurité peut être déclenchée. Ceci peut sembler logique car les données sont en théorie attachées à un domaine.

# JavaScript - Introduction

- Comment l'uiliser ?
  - les méthodes d'accès sont communes :
    - setItem(clé,valeur) : stocke une paire clé/valeur
    - getItem(clé) : retourne la valeur associée à une clé
    - removeItem(clé) : supprime la paire clé/valeur en indiquant le nom de la clé
    - key(index): retourne la clé stockée à l'index spécifié
    - clear(): efface toutes les paires
    - la propriété .length renvoie le nombre de paires stockées.

NB : Note : Les exemples suivants se basent sur sessionStorage mais fonctionneront de la même façon avec localStorage

# JavaScript - Introduction

- **Stockage**

```
sessionStorage.setItem("couleur","vert") ;
```

Le premier argument de setItem est la clé (toujours de type String). Elle précise l'endroit où sont stockées les données afin de pouvoir les y retrouver ultérieurement.

- **Récupération**

```
var couleur = sessionStorage.getItem("couleur") ;
```

Grâce à la clé initialement créée avec setItem il est possible de récupérer facilement les données. Ces dernières sont renvoyées sous la forme d'une String.

- **Suppression**

```
sessionStorage.removeItem("couleur") ;
```

Nous supprimons l'élément de session "couleur".

# JavaScript - Introduction

- **Suppression totale**

```
sessionStorage.clear() ;
```

Suppression complète de toutes les valeurs de session.

- **Accès direct**

Dans la plupart des situations, les variables seront accessibles directement en tant que membres de l'interface.

```
sessionStorage.couleur = "vert";
console.log(sessionStorage.couleur);
```

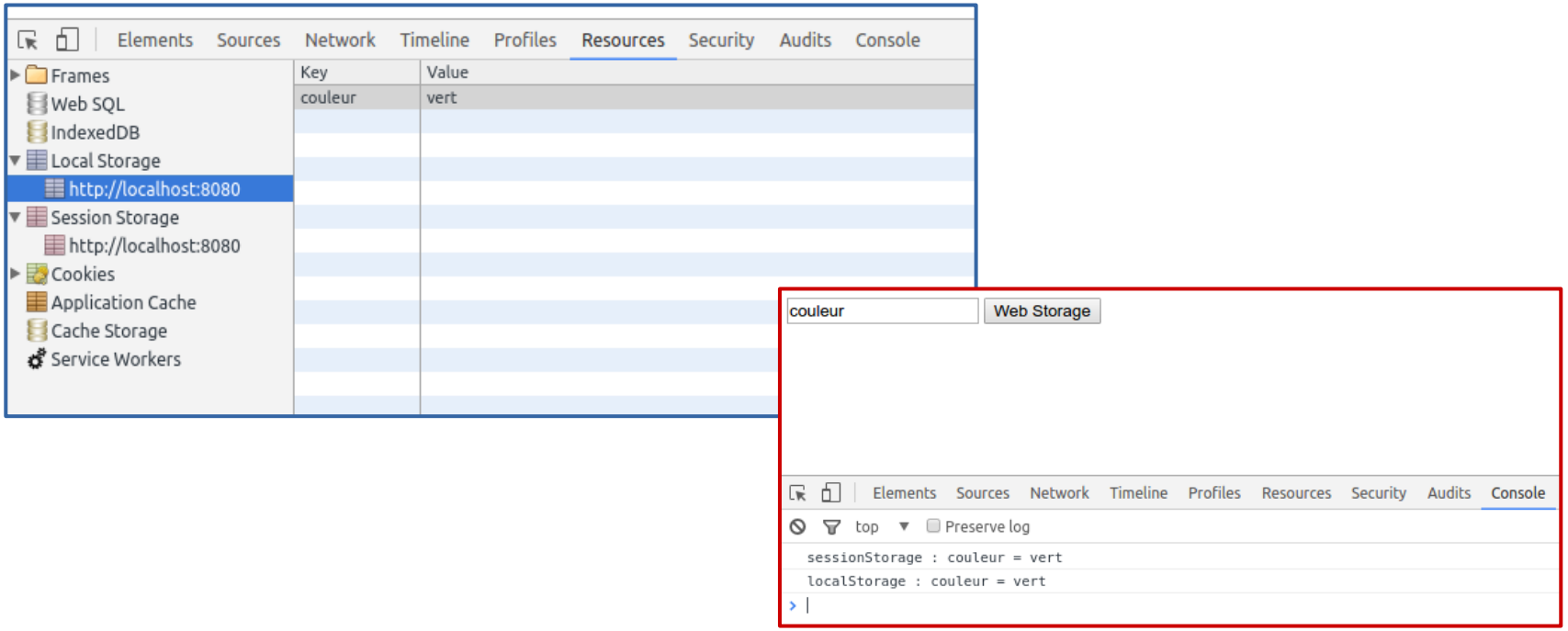
- **Examen et espace alloué**

Par défaut, Internet Explorer alloue 10 Mo par espace de stockage, et les autres navigateurs (Firefox, Chrome, Opera, Safari) 5 Mo par domaine. En réalité, par mesure de sécurité, on considère l'origine de la page, c'est-à-dire que sont également distingués tous les sous-domaines, ainsi que le port ou le mode de connexion (HTTPS s'il y a lieu). Une exception QUOTA_EXCEEDED_ERR est déclenchée si le quota est atteint.

# JavaScript - Introduction

- **Les outils de développement**

  Les outils de développement peuvent donner accès à des vues détaillées du stockage. Par exemple sous Chrome, l'onglet Resources :



# JavaScript - Introduction

- Du code vite !
  - Le fichier webstorage.html

```
<!DOCTYPE html>
<html>
<head>
  <title>webStorage exemple</title>
  <script src="js/webstorage.js"></script>
</head>
<body onload="writeStorage('couleur', 'vert')">
  <input type="text" name="key" id="key" value="couleur">
  <button onclick="readStorage()">Web Storage</button>
</body>
</html>
```

  - Le fichier webstorage.js

```
function writeStorage(key, value) {
  sessionStorage.setItem(key,value);
  localStorage.setItem(key, value);
}
function readStorage() {
  var key = document.getElementById('key').value;
  console.log('sessionStorage : '+ key +' = ' + sessionStorage.getItem(key));
  console.log('localStorage : '+ key +' = ' + localStorage.getItem(key));
}
```

# JavaScript - Introduction

- **Utilisation de JSON avec le WebStorage**
  - Web Storage est bien pratique pour stocker de simples String. Lorsqu'il s'agit de manipuler des données plus complexes, entre autres des objets JavaScript, il faut les linéariser au préalable en chaînes de texte puisque Web Storage n'accepte que ce type de données.
  - Le format **JSON** (JavaScript Object Notation) est la solution de prédilection. Deux méthodes équipent les navigateurs modernes : JSON.stringify() qui prend en paramètre un objet et renvoie une chaîne de texte linéarisée, et son inverse JSON.parse() qui reforme un objet à partir de la chaîne linéarisée.

# JavaScript - Introduction

- **Stockage json exemple :**
  - Stockage

```
var monobjet  = {
  propriete1 : "valeur1",
  propriete2 : "valeur2"
};
var monobjet_json = JSON.stringify(monobjet);
sessionStorage.setItem("monobjet",monobjet_json);
```

  - Lecture

```
var monobjet_json = sessionStorage.getItem("objet");
var monobjet = JSON.parse(monobjet_json);
// Affichage dans la console
console.log(monobjet);
```

Il serait possible d'exploiter le format JSON de façon beaucoup plus complexe et évoluée en fonction de la quantité de données à stocker et de leur provenance.

# JavaScript - Introduction

- *TP : Ecrire un programme JavaScript qui permet de stocker localemement la liste des pays récupérés par un appel Ajax pour la copier dans le localStorage.*
  - *Vous utiliserez des <button> comme déclencheurs pour l'appel ajax qui à la fin de son appel fera l'insertion dans le base Web de votre navigateur.*
  - *On fera une lecture de la localStorage pour récupérer les données et créer une liste déroulante.*

*NB : on utilise en général un test avant d'appeler l'objet sessionStorage ou localStorage. Trouvez lequel et utilisez le dans votre code.*

# JavaScript - Introduction

- **La gestion des événements en JavaScript**
  - JavaScript est un langage événementiel : le développeur a un contrôle limité sur le flux d'exécution du code, qui est déterminé principalement par les interactions avec l'environnement (activation d'un lien, mouvement de la souris, chargement du contenu du document, …).
  - La gestion des événements est un sujet essentiel dans le cadre de ce langage.
  - Cet partie présente les trois grandes familles d'interfaces qui sont aujourd'hui à notre disposition pour les évènements

# JavaScript - Introduction

- Le DOM niveau 0, standard de facto hérité de Netscape ; il s'agit de l'interface la plus largement supportée mais aussi la moins puissante.

- Le modèle d'événement DOM niveau 2

- L'interface spécifique à Internet Explorer, qui fournit un sous-ensemble des fonctionnalités du DOM niveau 2, mais avec des dénominations différentes.

# JavaScript - Introduction

- **L'objet Event**
  - Un événement est un changement d'état de l'environnement qui peut être intercepté par le code JavaScript.
  - Il est possible d'empêcher que l'activation d'un lien entraîne la navigation vers l'URL associée.
  - Ce changement d'état peut être provoqué par l'utilisateur (pression d'une touche, ...), par le document (chargement d'une image, ...), ou même par le développeur.
  - Cet objet, qui va se propager dans l'arbre DOM selon le flux d'événement contient des données dont certaines sont spécifiques au type d'événement (code touche keydown). Nous nous intéresserons ici aux données communes à tous les types d'événements, et seulement aux plus utilisées.

# JavaScript - Introduction

- **Propriétés ou méthodes de l'objet Event :**
  - **target** :
    - Il s'agit de la cible de l'événement. C'est le nœud de l'arbre DOM concerné par le changement d'état associé à l'événement. Dans le cas d'un événement de souris par exemple, c'est l'élément le plus profond de l'arbre au-dessus duquel se trouve la souris.

```
// event est l'objet Event
var target = event.target
```

  - **type** :
    - Comme son nom l'indique, c'est le type d'événement ("focus", "load", ...).

# JavaScript - Introduction

- **stopPropagation** :
  - Cette méthode permet d'arrêter la propagation de l'événement dans l'arbre DOM après le nœud sur lequel il se trouve.

```
// event est l'objet Event
if (event.stopPropagation) {
  event.stopPropagation();
}
```

  - On utilise en général cette méthode en parallèle avec preventDefault.

- **PreventDefault :**
  - Pour les types d'événements qui l'autorisent, il est possible grâce à cette méthode d'annuler l'action implicite correspondante. Par exemple, l'action implicite associée à un événement de type submit est l'envoi au serveur du formulaire concerné.

# JavaScript - Introduction

- Pour un gestionnaire d'événement DOM-0, on se contente en général du code « return false; », car, bien que non défini dans la spécification, il est beaucoup plus largement supporté.

```
// event est l'objet Event
if (event.preventDefault) {
  event.preventDefault();
} else return false ;
```

- **currentTarget** :
  - Il s'agit du nœud sur lequel l'événement se trouve actuellement dans le cadre du flux d'événement.
  - Il est à noter que, contrairement aux données précédentes, celle-ci change au cours de la progression de l'événement dans l'arbre DOM.

# JavaScript - Introduction

- **Le flux d'événement**
  - Une fois l'objet Event créé, il se propage dans l'arbre DOM selon un flux bien précis déterminé par sa cible.
    - Phase de capture : l'événement se propage de la racine du document (incluse) à la cible (exclue).
    - L'événement atteint la cible.
    - Phase de bouillonnement (bubbling) : l'événement se propage dans le sens inverse : de la cible (exclue) à la racine du document (incluse). Par exemple, pour la page HTML suivante :

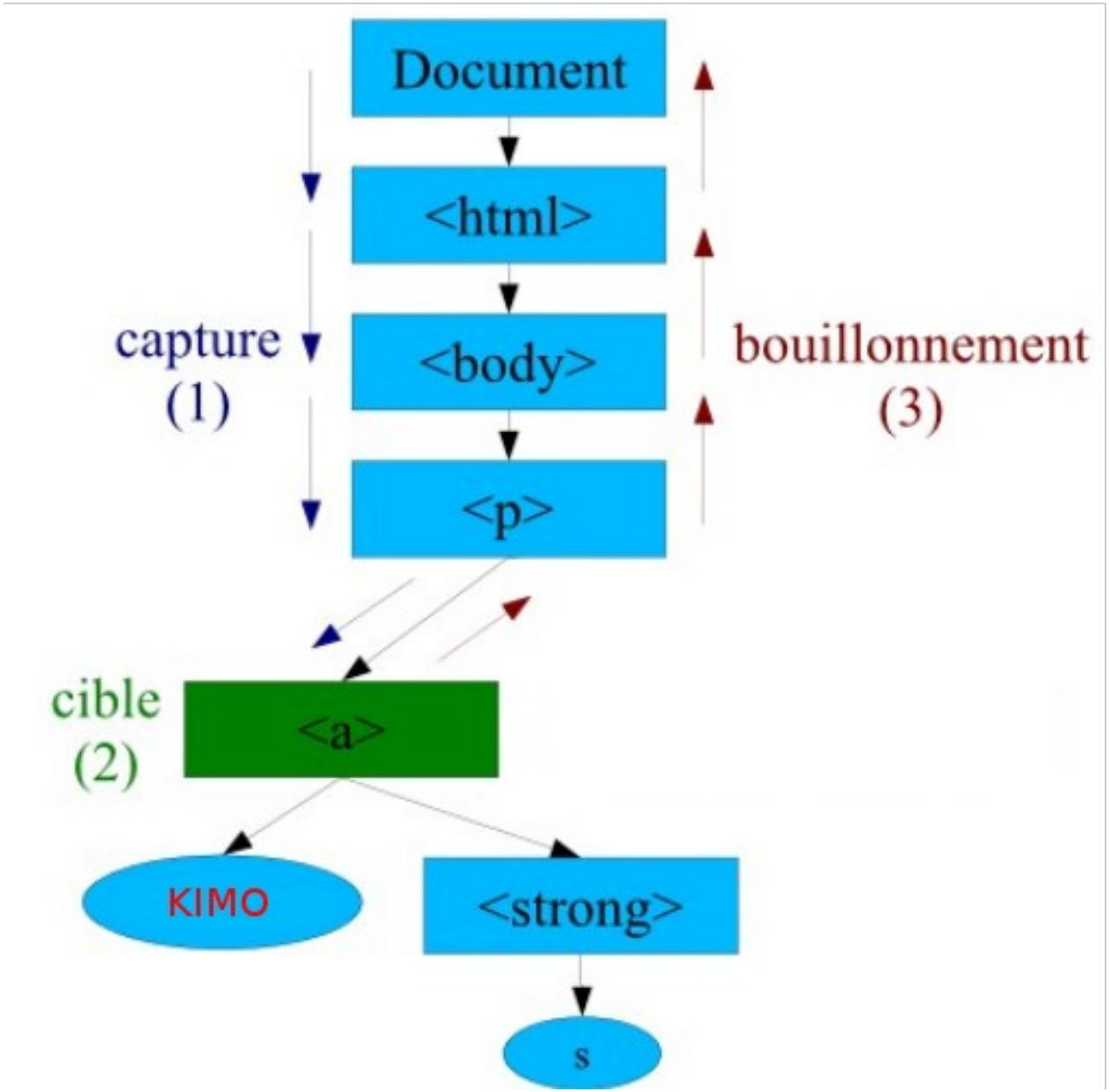
```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8" />
  <title>Event exemple</title>
</head><body><p>
    <a href="http://www.kimo.fr/">Kimo<strong>s</strong></a>
  </p>
</body></html>
```

# JavaScript - Introduction

- **Si l'utilisateur active le lien,**
  - voici les différentes étapes qui s'enchaînent :
    - Un événement de type click est créé. La cible est déterminée ; il peut s'agir de a comme de strong (dans le cas où l'utilisateur utilise une souris et a cliqué sur la zone correspondant à la lettre « s »). Supposons que la cible est l'élément a.
    - Phase 1 (capture) : l'événement se propage du nœud Document (inclus) au nœud a (exclu).
    - Phase 2 (cible) : l'événement atteint le nœud a.
    - Phase 3 (bouillonnement) : l'événement se propage du nœud a (exclu) au nœud Document (inclus).

Voyons un schéma de cette propagation

# JavaScript - Introduction




La phase de bouillonnement est optionnelle. Son existence est déterminée par le type d'événement : par exemple, les événements de type load ne bouillonnent pas.

De plus, comme expliqué dans la partie précédente, le flux d'événement peut être interrompu dans le code JavaScript grâce à la méthode stopPropagation.

# JavaScript - Introduction

- **Les gestionnaires d'événement**
  - Pour pouvoir intercepter un événement pendant sa propagation dans l'arbre DOM, il faut utiliser un gestionnaire d'événement. En JavaScript, il s'agit d'une fonction que l'on attache à un nœud, en précisant :
    - un type d'événement,
    - une phase : capture, ou bouillonnement (voir la partie sur le flux d'événement), sachant que l'on inclut la cible dans la phase de bouillonnement et pas dans la phase de capture.
  - Un gestionnaire d'événement est appelé lorsqu'un événement atteint le nœud auquel il a été attaché si le type de l'événement et la phase actuelle du flux d'événement correspondent à ceux précisés lors de son ajout.

# JavaScript - Introduction

- **Les gestionnaires d'événement DOM-2**
  - Le modèle d'événement DOM niveau 2 définit deux méthodes, addEventListener et removeEventListener, qui permettent d'attacher ou de détacher un gestionnaire d'événement d'un nœud :

```
element.addEventListener(type, listener, useCapture);

element.removeEventListener(type, listener, useCapture) ;
```

    - type est le type d'événement,
    - listener le gestionnaire d'événement (en JavaScript, une fonction),
    - useCapture un booléen : true pour la phase de capture, ou false pour la phase de bouillonnement et la cible. On utilise quasiment toujours la valeur false.

# JavaScript - Introduction

- **Voici un exemple d'utilisation de addEventListener :**

```
function envoiForm(event) {
  if (this.elements.adresse.value === "") {
    alert("L'adresse est vide.");
    event.preventDefault();
  }
}
var formulaire = document.getElementById("coordonnees");
formulaire.addEventListener("submit", envoiForm, false);
```

  - On suppose qu'il existe au moment de l'exécution de ce code un formulaire d'id « coordonnees » qui contient un champ dont le nom (name) est « adresse ».
  - Lorsque ce formulaire sera envoyé, l'événement de type submit sera intercepté par le gestionnaire d'événement associé à la fonction envoiForm. Si le champ « adresse » est vide, un dialogue d'alerte sera affiché et l'envoi du formulaire sera annulé (preventDefault).

# JavaScript - Introduction

- La fonction qui fait office de gestionnaire d'événement a accès à l'objet Event par l'intermédiaire de son premier paramètre (ici event). De plus, le nœud auquel le gestionnaire d'événement a été ajouté (c'est-à-dire event.currentTarget) est associé à this (comportement non spécifié par le modèle d'événement DOM niveau 2, mais très homogène sur les différentes implémentations), ce dont on se sert ici pour récupérer le champ « adresse » (this.elements.adresse).
- Pour détacher un gestionnaire d'événement, il faut passer à la méthode removeEventListener les mêmes paramètres que ceux qui ont été passés à addEventListener. Par exemple :

```
formulaire.removeEventListener("submit", envoiForm, false);
```

- Ces deux méthodes permettent donc de manipuler plusieurs gestionnaires d'événement de même type sur un même nœud.

# JavaScript - Introduction

- **Les gestionnaires d'événement DOM-0**
  - Il s'agit en fait d'un standard de facto, hérité de Netscape, mais qui est encore aujourd'hui la façon la plus fiable de gérer les événements en JavaScript.
  - Il ne supporte que la phase de bouillonnement (et la cible), et ne permet pas d'ajouter plusieurs gestionnaires d'événement de même type à un même nœud.
  - Pour ajouter un gestionnaire d'événement à un élément, il suffit de définir une fonction comme propriété de l'objet JavaScript correspondant. Par exemple :

# JavaScript - Introduction

- **DOM-0 Exemple :**

```
function envoiForm(event) {
  if (this.elements.adresse.value === "") {
    alert("L'adresse est vide.");
    return false;
  }
}
var formulaire = document.getElementById("coordonnees");
formulaire.onsubmit = envoiForm;
```

Une autre façon d'utiliser les gestionnaires d'événements DOM-0 (et sans doute la plus utilisée) est de passer par les attributs HTML correspondants.

```
<form id="coordonnees"
  onsubmit="alert(event.type);"
  action="traiter-coordonnees" method="post">
  (…)
```

Une autre façon d'utiliser les gestionnaires d'événements DOM-0 (et sans doute la plus utilisée) est de passer par les attributs HTML correspondants.

```
formulaire.onsubmit = function(event) {
  alert(event.type);
};
```

# JavaScript - Introduction

- L'utilisation des gestionnaires d'événements par l'intermédiaire des attributs HTML est en général à éviter, car elle va à l'encontre de la sépararation du comportement et de la structure.

- **Conclusion**
  - Les gestionnaires d'événement DOM-0 représentent encore le moyen le plus facile de manipuler les événements en JavaScript, notamment parce qu'ils ne nécessitent pas plusieurs branches de code en ce qui concerne l'ajout ou la suppression des gestionnaires d'événement,

# JavaScript - Introduction

- *TP : Ajouter à vote formulaire de contact un evènement permettant d'intercepter l'évènement « submit » et de vérirfier que la valeur du nom est conforme (édictons une règle juste pour le jeu : longueur de la chaîne de caractère inférieure à 25 lettres).*

# JavaScript - Introduction

- **Elément de correction :** fichier HTML contacts

```
<!DOCTYPE html>
<html>
<head>
  <title>event exemple</title>
  <link rel="stylesheet" href="css/event.css" media="screen" />
  <script src="js/event.js"></script>
</head>
  <body onload="init()">
    <form name="coordonnees" id="coordonnees" method="POST" action="#">
      <fieldset>
        <legend>Informations personnelles </legend>
             (...)
        <label> for="prenom">Prénom : </label>
        <input type="text" name="prenom" id="prenom"
               placeholder="Votre Prénom" required/><br/><br/>
        <label for="adresse">Adresse : </label>
        <input type="text" name="adresse" id="adresse"
               placeholder="Votre Adresse"/><br/><br/>
             (...)
        <input type="submit" value="Valider" />
      </fieldset>
    </form>
  </body>
</html>
```

# JavaScript - Introduction

- Code JavaScript :

```
var formulaire;

function envoiForm(event) {
  if (formulaire.elements.adresse.value === "") {
    alert("L'adresse est vide.");
    event.preventDefault();
  }
}

function init() {
  formulaire = document.getElementById("coordonnees");
  formulaire.addEventListener("submit", envoiForm, false);
}
```

# Jquery : Une librairie pratique