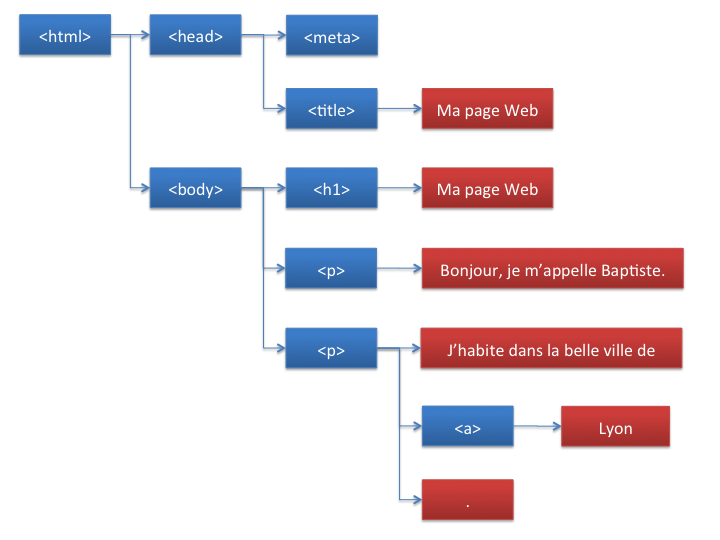
**Créez des pages web interactives avec Javascript**

**Introduction au DOM**

Le DOM représente une page web sous la forme d'une hiérarchie d'objets, où chaque objet correspond à un nœud de l'arborescence de la page. Les objets du DOM disposent de propriétés et de méthodes permettant de les manipuler avec JavaScript.

La structure d’une page Web



Chaque entité de l'arborescence est appelée un **nœud**(node). On distingue deux types de nœuds :

* Ceux (en bleu) qui correspondent à des éléments HTML, comme <body> ou<p>. Ces nœuds peuvent avoir des sous-nœuds, appelés **fils** ou **enfants**(children).
* Ceux (en rouge) qui correspondent au contenu textuel de la page. Ces nœuds ne peuvent pas avoir de fils.

**Premiers pas avec le DOM**

Il existe une variable document qui permet d’accéder à n’importe quel nœud.

Cette variable est un objet et dispose des propriétés head et body qui permettent d'accéder respectivement aux éléments<head> et <body> de la page.

**Découvrir le type d'un nœud**

Chaque objet du DOM a une propriété nodeType qui indique son type. La valeur de cette propriété est document.ELEMENT\_NODE pour un nœud "élément" (balise HTML) et document.TEXT\_NODE pour un nœud textuel.

Chaque objet du DOM de type élément possède une propriété childNodes. Il s'agit d'une collection ordonnée regroupant tous ses nœuds enfants sous la forme d'objets DOM. On peut utiliser cette collection un peu comme un tableau pour accéder aux différents enfants d'un nœud.

// Affiche les noeuds enfant du noeud body

for (*var* i = 0; i < document.body.childNodes.length; i++) {

*console*.log(document.body.childNodes[i]);

}

#### Accéder au parent d'un nœud

Chaque objet du DOM possède une propriété parentNode qui renvoie son nœud parent sous la forme d'un objet DOM.

*var* h1 = document.body.childNodes[1];

*console*.log(h1.parentNode); // Affiche le noeud body

*console*.log(document.parentNode); // Affiche null : document n'a aucun noeud parent

**Sélectionner des éléments**

**Sélection d’éléments selon leur balise**

Tous les éléments du DOM possèdent une méthode getElementsByTagName. Celle-ci renvoie une liste des éléments qui portent le nom de la balise qui a été passée en paramètre lors de l'appel.

Si par exemple je souhaite récupérer les h2 j’écris :

var titre = document.getElementByTagName("h2") ;

**Sélection d’éléments selon leur class**

Les éléments du DOM disposent également d'une méthode getElementsByClassName. Elle renvoie une liste des éléments ayant le nom de classe passé en paramètre. Là encore, la recherche concerne l'ensemble des sous-éléments du nœud sur laquelle la méthode est appelée, exemple :

<ul class="merveilles">

var elts = document.getElementByClassName("merveilles") ;

Pour sélectionner et afficher l'ensemble des éléments du document ayant pour classe "merveilles", vous pouvez écrire le code ci-dessous.﻿﻿

// Tous les éléments ayant la classe "merveilles"

*var* merveillesElts = document.getElementsByClassName("merveilles");

for (*var* i = 0; i < merveillesElts.length; i++) {

*console*.log(merveillesElts[i]);

}

**Sélection d’éléments selon leur id**

Pour récupérer un élément j’utilise la méthode : document.getElementById() et je mets en paramètre l’id souhaité.

Si je veux récupérer tous les éléments ayant une class il faut écrire : document.getElementsByClassName() et mettre entre paramètres les différentes classes que je souhaite récupérer.

**Sélection d’éléments à partir d’un sélecteur CSS**

Exemple : imaginons qu'on souhaite obtenir toutes les éléments <li> correspondant à des merveilles antiques qui existent toujours. Elles sont identifiées par la classe "existe" dans la page web. Pour les sélectionner, on pourrait écrire le code ci-dessous.﻿﻿

// Tous les éléments fils de l'élément d'identifiant "antiques" ayant la classe "existe"

*console*.log(document.getElementById("antiques").getElementsByClassName("existe").length); // Affiche 1

Cependant, cette syntaxe est un peu lourde et sujette aux erreurs.

Pour faciliter la sélection d'éléments suivant des critères complexes, le DOM s'est enrichi de deux nouvelles méthodes. La première, nommée querySelectorAll, permet de rechercher des éléments à partir d'un **sélecteur CSS**.

Grâce à cette méthode, on peut rechercher des éléments du DOM en définissant un sélecteur identique à celui qu'on utiliserait dans une feuille de style CSS. Elle renvoie la liste des éléments correspondant au sélecteur passé en paramètre.

Voici quelques exemples des possibilités offertes par cette méthode.

// Tous les paragraphes

*console*.log(document.querySelectorAll("p").length); // Affiche 3

// Tous les paragraphes à l'intérieur de l'élément identifié par "contenu"

*console*.log(document.querySelectorAll("#contenu p").length); // Affiche 2

// Tous les éléments ayant la classe "existe"

*console*.log(document.querySelectorAll(".existe").length); // Affiche 8

Voici comment on peut utiliser querySelectorAll pour sélectionner les merveilles antiques qui existent toujours.

// Tous les éléments fils de l'élément identifié par "antiques" ayant la classe "existe"

*console*.log(document.querySelectorAll("#antiques > .existe").length); // Affiche 1

L'autre méthode de recherche d'éléments à partir d'un sélecteur CSS s'appelle querySelector. Elle fonctionne de manière identique à querySelectorAll, mais renvoie uniquement le premier élément correspondant au sélecteur passé en paramètre.﻿﻿

// Le premier paragraphe

*console*.log(document.querySelector("p"));