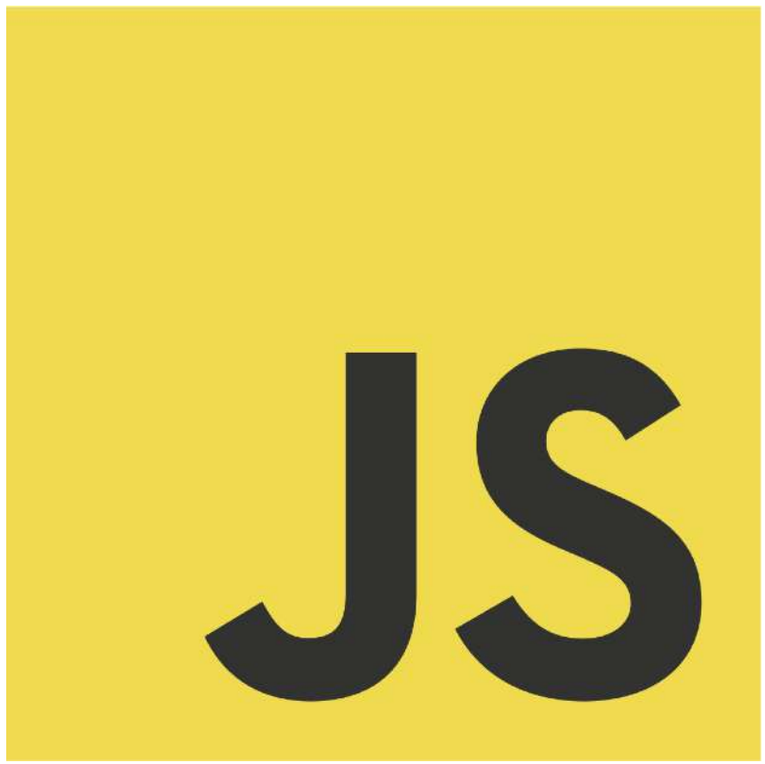


M4103C - Programmation Web Orientée Client

C1 – Le langage JavaScript

2019 - Carl VINCENT (carl.vincent@iut2.univ-grenoble-alpes.fr)
(Basé sur le cours de Christophe BROUARD)

ORGANISATION DU MODULE

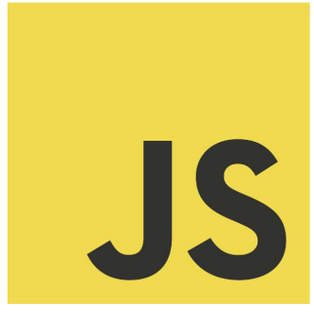
Numero semaine / [numero calendaire]		Cours	TD
semaine 1 [7]	JavaScript & DOM	CM 1.5h A	TD 2h SX
semaine 2 [9]			TD 2h SX TD 2h SX
semaine 3 [10]	AJAX & JQuery	CM 1.5h A	TD 2h SX
semaine 4 [11]			TD 2h SX TD 2h SX
semaine 5 [12]	Présentation Projet + Frameworks JS	CM 1.5h A	TD 2h SX
semaine 6 [13]	(Projet)		TD 2h SX TD 2h SX
semaine 7 [14]	(Projet)		TD 2h SX TD 2h SX

Coefficient global : 15

Contrôle machine : 10

Projet : 5

SOMMAIRE DE CE 1ER COURS



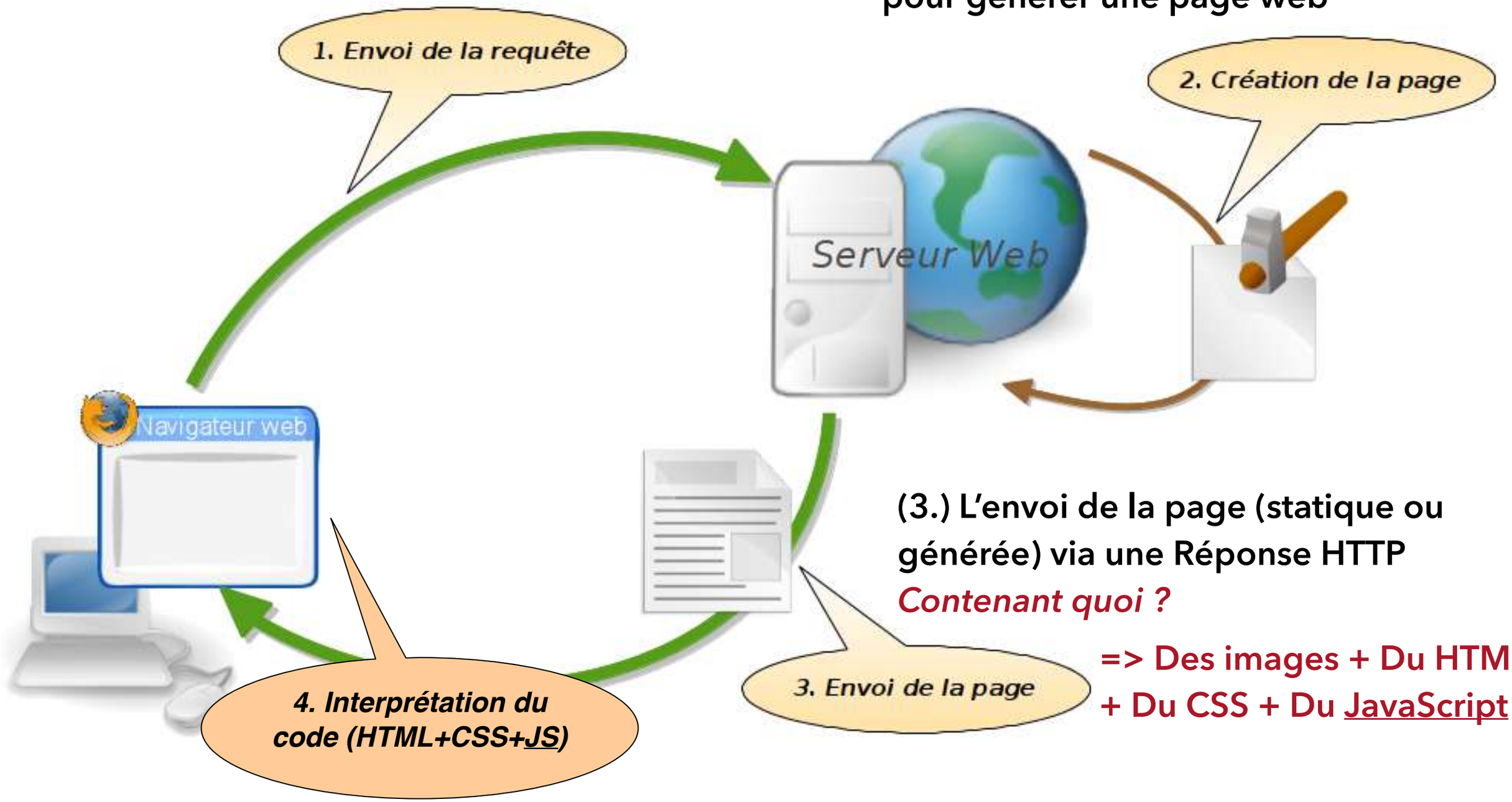
1. Le JavaScript : Qui ? Quoi ? Pourquoi ? Comment ?
2. L'intégration du JavaScript dans une page web
3. Les bases du JavaScript (variables, opérateurs, fonctions, …)
4. Les tableaux et objets
5. Le DOM et l'objet "window"
6. La manipulation du DOM et la gestion des évènements

Mais avant de se lancer, revoyons les bases …

QU'EST-CE QU'IL SE PASSE QUAND ON ACCÈDE À UNE PAGE WEB ?

(1.) L'envoi d'une requête HTTP (GET ou POST)

(2.) L'envoi d'une page web statique OU l'exécution d'un script PHP/Ruby/… pour générer une page web

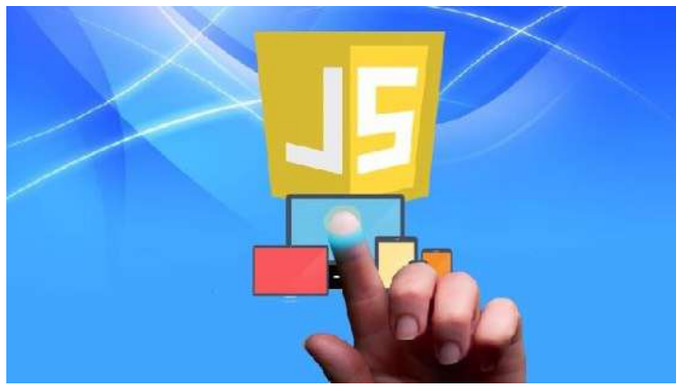


(3.) L'envoi de la page (statique ou générée) via une Réponse HTTP

Contenant quoi ?

=> Des images + Du HTML + Du CSS + Du JavaScript !

Pourquoi a été créé le JavaScript ?



- À la base, le JavaScript a surtout été créé pour **interagir avec le HTML d'une page Web** pour rendre cette page **dynamique :**
  - ➡ **Interagir :** savoir qu'un bouton a été cliqué, qu'une donnée a été mise à jour, vérifier le champ d'un formulaire, …
  - ➡ **Afficher / Masquer :** rendre visible / masquer des parties de la page, gérer des animations, …
  - ➡ **Communiquer :** envoyer ou recevoir des requêtes (HTTP ou Base de données) via une requête AJAX (Asynchronous JavaScript And XML)
- L'un des buts était de **réduire les appels au serveur Web** pour accélérer le contrôle des ressources du client (documents, formulaires, …)
  => On parle de "**Client Riche**"

Le JavaScript, c'est quoi ?

- Le JavaScript (souvent appelé « JS ») un **langage de programmation** de **script** et **orienté objet**
  - ➡ Langage de **script** interprété à **typage dynamique** = un langage de **haut-niveau** qui a besoin d'un **interpréteur** pour s'exécuter

Hello-World.js

```
var msg = "Hello world!";
alert(msg);
```

  - ➡ Langage **orienté objet** dit "à prototype" = **En JavaScript, (quasiment) tout est objet !** Un objet JS est un simple clone d'un « objet de base » appelé « **Prototype** » (et non pas l'instance d'une classe)

Everything-is-an-object.js

```
var monObjet = {};
monObjet.texte = "Un peu de texte";
console.log(monObjet) // {texte: "Un peu de texte"}
console.log(monObjet.texte.length); // 15
```

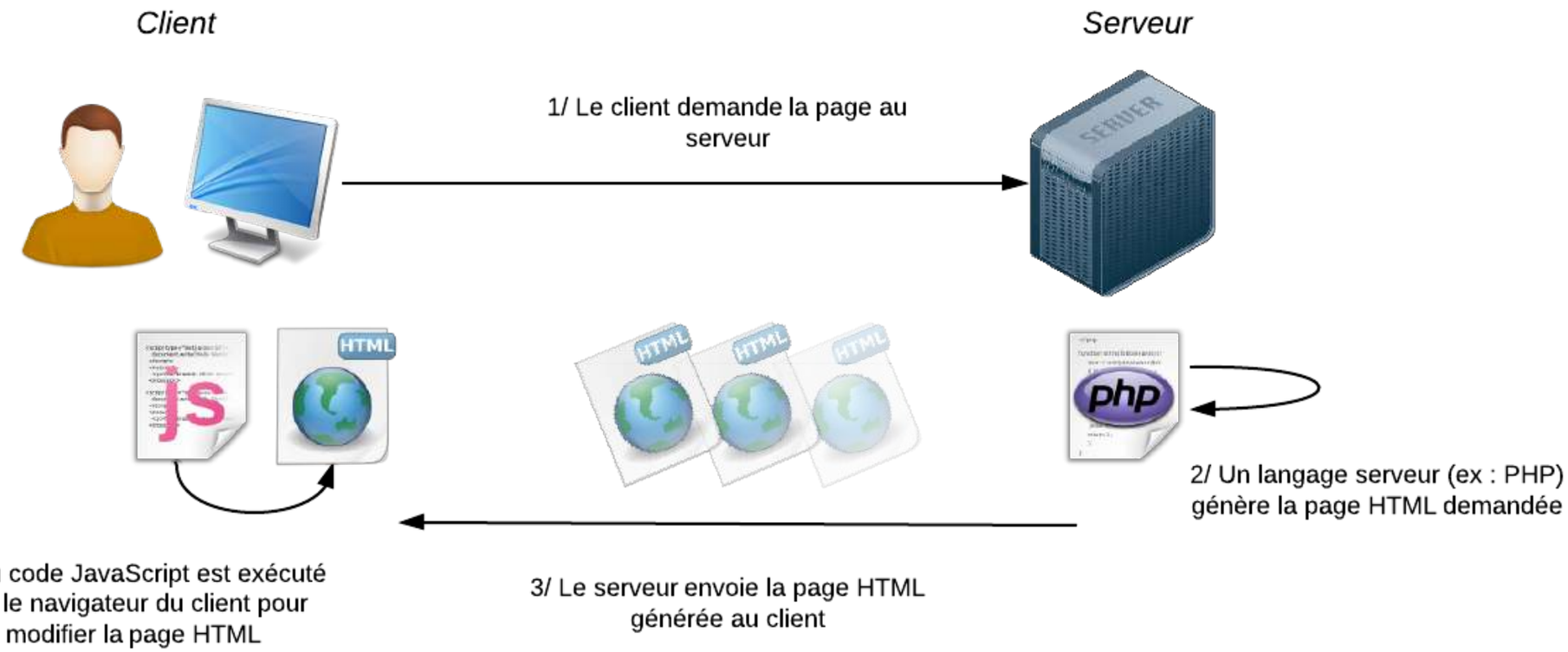
Qui est derrière le JavaScript ?

Brendan Eich en 2012

- Création du JavaScript en **1995** par **Brendan Eich**, un ingénieur américain de **Netscape**
- **Brendan Eich** participera ensuite à la création de la **Mozilla Foundation**
- Au départ, le langage devait s'appeler **LiveScript** mais le nom fut changé en **JavaScript** pour **capitaliser sur la popularité du langage Java** (/!\ Mais aujourd'hui, les langages Java & JavaScript n'ont rien à voir !)
- Aujourd'hui, le JavaScript est un **langage standardisé** par l'**ECMA** et il est maintenant géré par tous les navigateurs modernes : http://www.ecma-international.org/ecma-262/

COMMENT FONCTIONNE LE JS AU SEIN D'UNE PAGE WEB ?

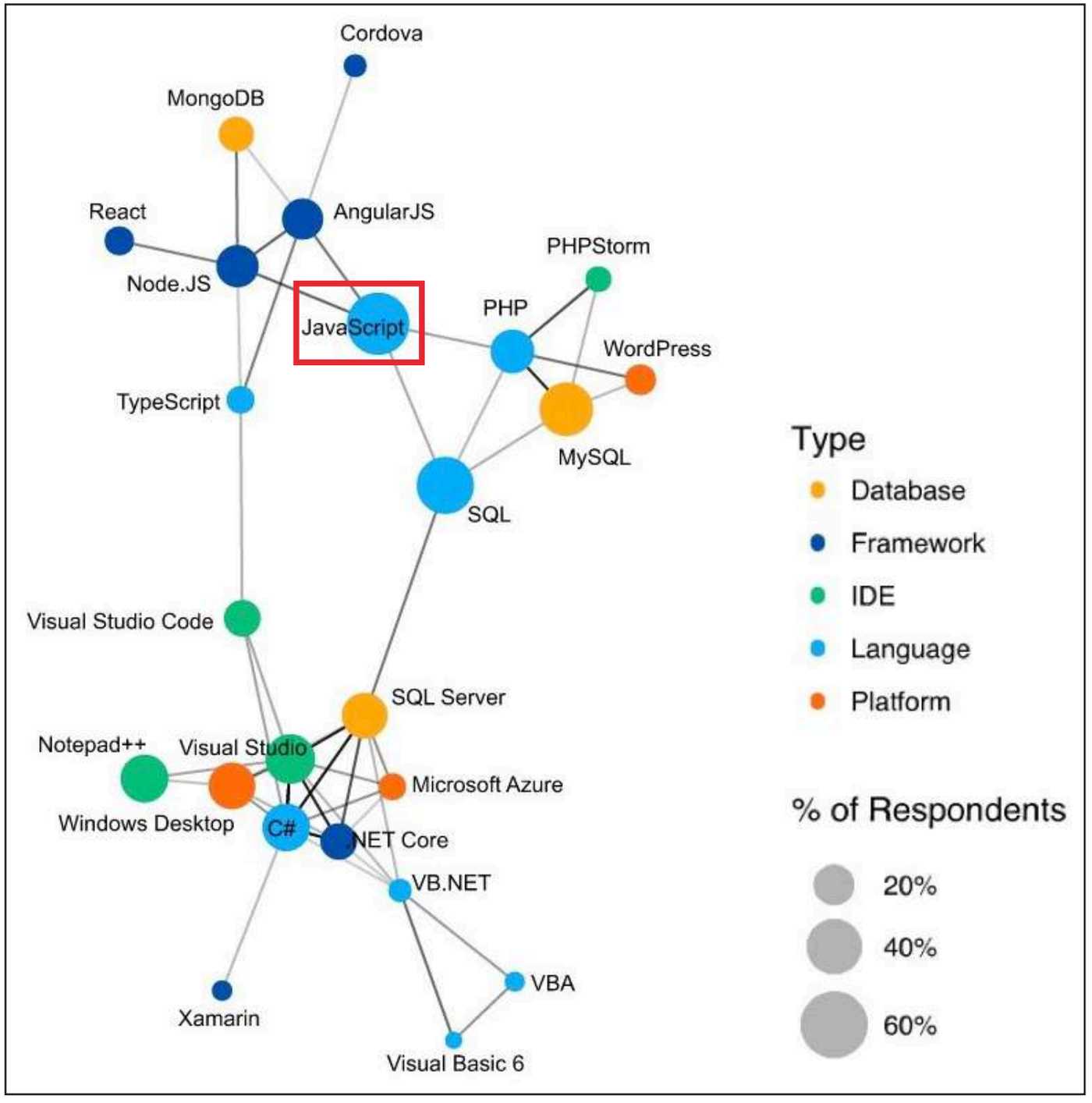
- Le JavaScript est exécuté par **l'interpréteur** du navigateur web (aussi appelé « Machine virtuelle » ou « Moteur JS »)
  => On dit que le JS est exécuté **côté client**



- Le JS interagit avec une page Web via le **DOM** (**Document Object Model**) qui est **une représentation structurée de la page web** sous forme d'un arbre

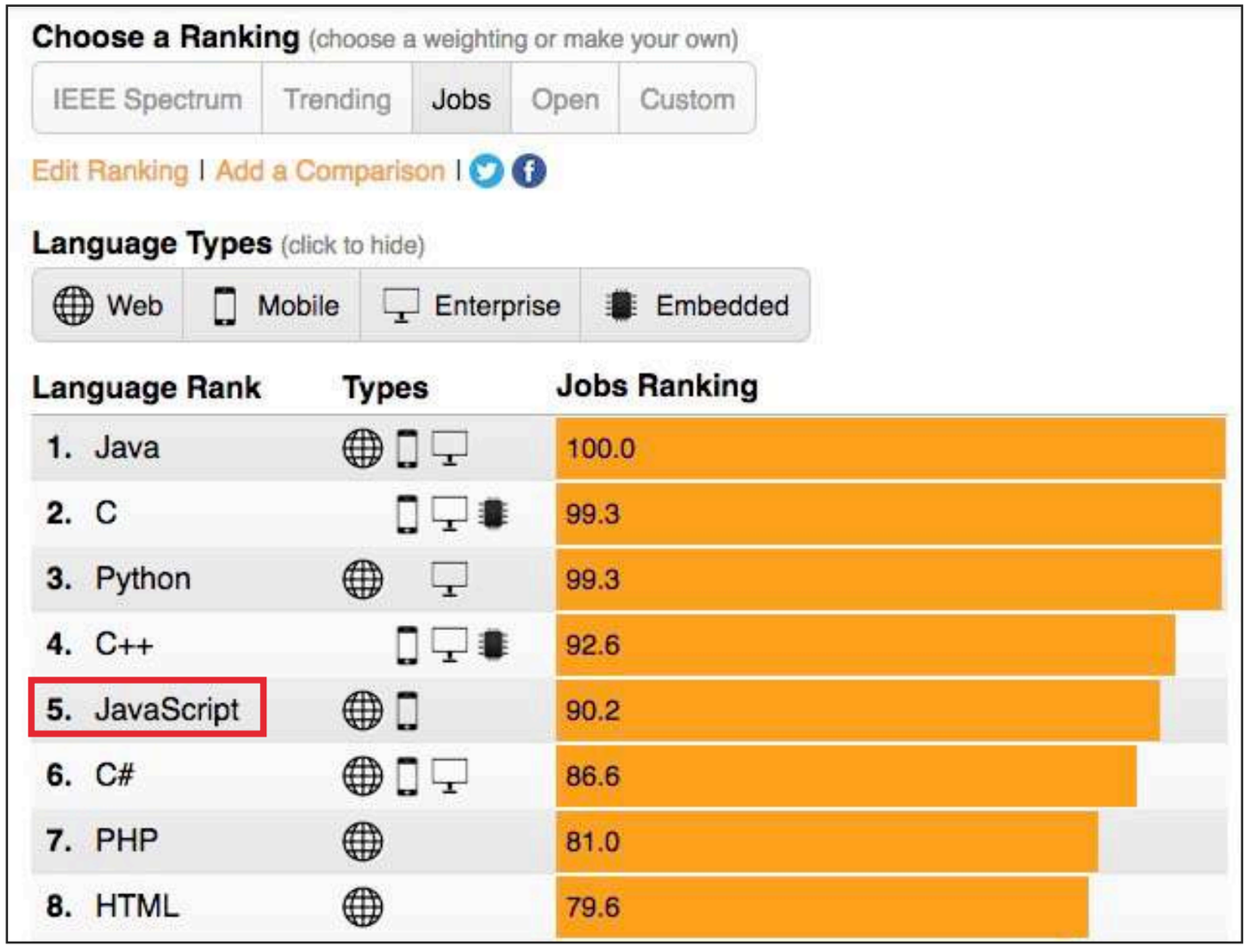
Le JavaScript de nos jours

- **Un incontournable du Web** avec des utilisations qui se diversifient



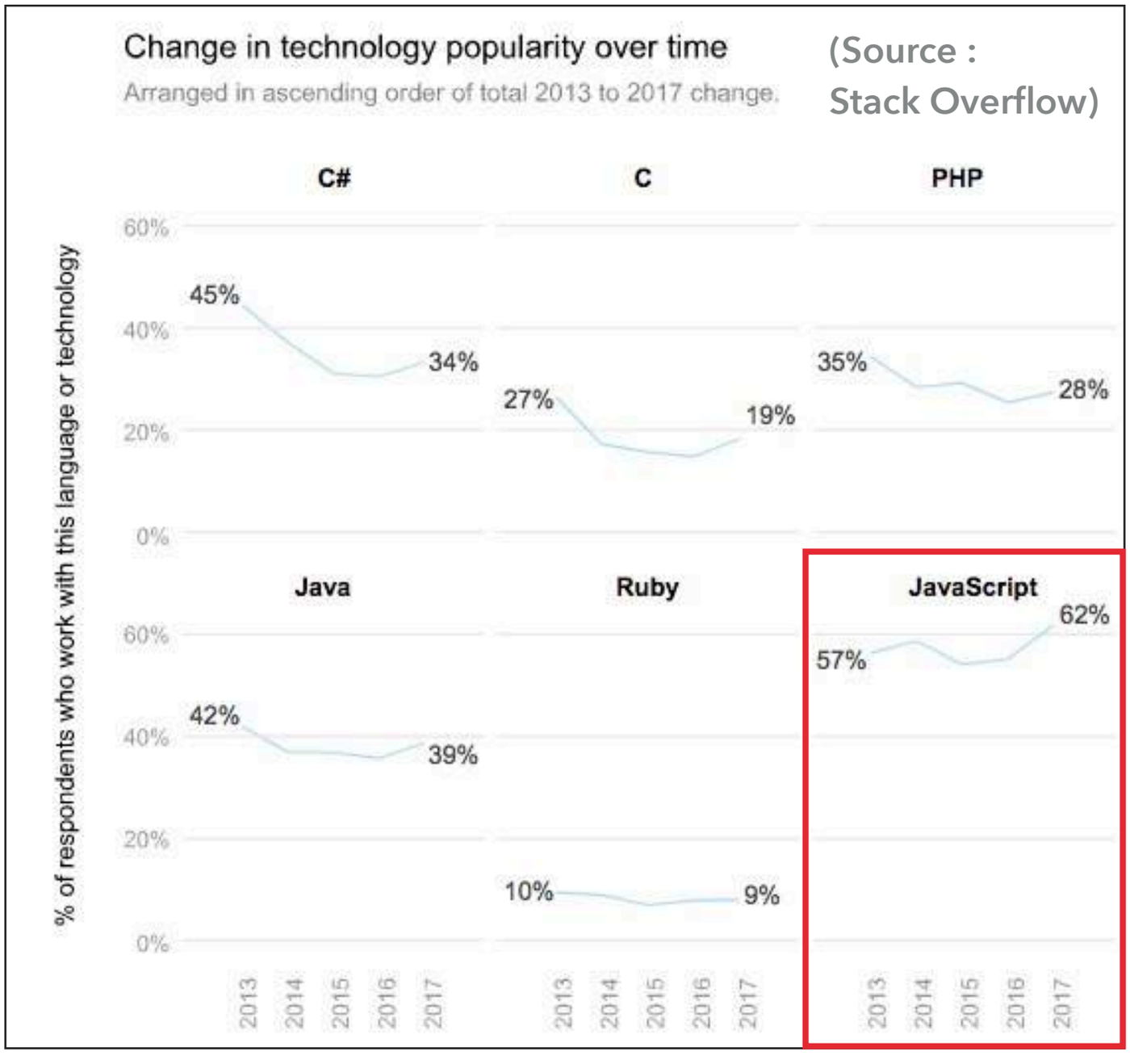
Le JavaScript de nos jours

- **Un incontournable du Web** avec des utilisations qui se diversifient
- **Une compétence** qui est de plus en plus demandée



Le JavaScript de nos jours

- **Un incontournable du Web** avec des utilisations qui se diversifient
- **Une compétence** qui est de plus en plus demandée
- **Une popularité élevée** ces dernières années



Insérer du JavaScript dans une page web : Du JS dans du HTML

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello !</h1>

    <!-- Balise <script> (La spécification HTML5 conseille
      d'omettre l'attribut : type="text/javascript") -->
    <script>
       // Initialisation de la variable msg
       var msg = "Hello world !";
       /* Affichage d'une boîte de dialogue avec le message */
       alert(msg);
    </script>
  </body>
</html>
```

Insérer du JavaScript dans une page web : Un fichier JS à part

Fichier "hello-world.html"

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello !</h1>

    <!-- Balise <script> avec attribut src
      (à placer à la fin du <body> pour ne pas
       retarder le chargement de la page) -->
    <script src="monScript.js"></script>
  </body>
</html>
```

Fichier "monScript.js"

```js
/* Affiche une boite de dialogue au chargement
de la page */
alert("Hello World !!");
```

=> Méthode préférable car cela sépare bien le HTML et le JS

Syntaxe JS : Les Variables et Types

- **Déclaration de variable** (≠ PHP) :

```
var num = 42;
var text = "Ceci est mon texte";
var vide, nonVide = 123;
var vrai = true, faux = false;
```

- **Typage dynamique** (= PHP) :

```
var maVariable = 42;
console.log(typeof maVariable);
// Affiche "number" dans la console
maVariable = 'C\'est ma variable';
console.log(typeof maVariable);
// Affiche "string" dans la console
maVariable = false;
console.log(typeof maVariable);
// Affiche "boolean" dans la console
```

Syntaxe JS : Les Opérateurs

- **Opérateurs arithmétiques** (/!\ La concaténation se fait avec un "**+**" contrairement au PHP /!\) :

```
var n1=4, n2=2, res1, res2;
res1 = n1 + n2; // res1 = 6
res2 = n1 / n2; // res2 = 2
res2 += 40;     // res2 = 42
res2 -= n1;     // res2 = 38
```

```
// +, -, *, /, %, +=, -=, ++, --, …
number += 5; // éq. à : number = number+5;
var text = 'Bonjour';
text += ' toi';
alert(text); // Affiche "Bonjour toi"
```

- **Opérateurs logiques et de comparaison** (= PHP) :

```
// ===, !== (compare valeurs ET types)
// ==, != (compare seulement les valeurs)
// >=, >, <, <=, !, &&, ||, ...
var number = 4, text = '4', res;
res = number == text; // res = true
res = number === text; // res = false
res = number >= text; // res = true
console.log(typeof res); // Affiche "boolean"
```

STRUCTURES DE CONTRÔLE : INSTRUCTIONS CONDITIONNELLES (1)

- Les conditions en "**if / else if / else**" sont classiques :

```
var val = parseInt(prompt("Entrez un
nombre :"));
if (val === 0) {
    alert("La valeur est nulle");
} else if(val > 0) {
    alert("La valeur est positive");
} else if(val < 0) {
    alert("La valeur est négative");
} else {
    alert("Ceci n'est pas un nombre !");
}
```

```
// Toute variable peut être
// interprétée comme un bool.
var cond = "Une variable est
considérée comme vrai, sauf si
elle est égale à 0, NaN, '',
null ou undefined";
if(cond){
  alert("Condition vérifiée");
}
```

- Les **conditions ternaires** peuvent remplacer les conditions en "**if / else**" :

```
var age = parseInt(prompt("Quel est votre âge ?"));
var categorie = (age >= 18) ? "majeur" : "mineur";
alert("Vous êtes " + categorie);
```

STRUCTURES DE CONTRÔLE : INSTRUCTIONS CONDITIONNELLES (2)

- L'instruction **typeof** permet de **tester l'existence** d'une variable :

```
if(typeof inconnue === "undefined"){
   alert("La variable est inconnue");
}
```

- Les conditions "**switch**" permettent de facilement tester une égalité entre une multitude de valeurs :

```
var jour = prompt("Quel jour on est ?");
switch (jour) {
  case "vendredi":
     alert("Yes ! C'est bientôt le week-end !");
     break;
  case "samedi":
  case "dimanche":
     alert("C'est le week-end ! Yeah !!!");
     break;
  default:
     alert("Bon c'est quand le week-end...");
}
```

Structures de contrôle : Boucles

```
var str = "";
for (var i = 0 ; i < 9 ; i++) {
  str = str + i;
}
console.log(str);
// Affiche : "012345678"
```

```
var n = 0;
while (n < 3) {
  n++;
}
console.log(n);
// Affiche : 3
```

```
var iter = 1;
do {
  console.log("Itération " + iter);
  iter++;
}
while (iter < 5);
// Affiche : "Itération 1" … "Itération 4"
```

FONCTIONS ET PORTÉE DE "VAR"

- Toute variable déclarée dans une fonction n'est utilisable **que dans cette même fonction !**
  –> c'est une **variable locale**

- Dans une fonction, **la variable locale prend le dessus sur la variable globale de même nom**

```
function myFunction(arguments) {
  /* Le code que la fonction
       va exécuter */
}
```

```
var msg = 'Hello world';
function sayHello() {
  alert(msg);
}
sayHello(); // 'Hello world'
// /!\ Fortement déconseillé !!!

function sayHello2() {
  var msg2 = 'Hello world';
}
sayHello2(); // (Rien n'apparait)
alert(msg2); // (Erreur !!)
```

```
var msg = 'Ici, variable ';
var scope = 'GLOBALE';
function showMsg() {
  var scope = 'locale';
  alert(msg + scope);
}
showMsg(); //'Ici, variable locale'
alert(msg + scope); //'Ici, variable GLOBALE'
```

FONCTIONS (SUITE)

- Il est possible d'avoir des **arguments facultatifs**
- Il est aussi possible d'avoir des arguments **avec une valeur par défaut**
- Une fonction **peut ne pas avoir de nom**, dans ce cas elle est appelée "**fonction anonyme**" (ces fonctions nous seront bien utiles par la suite)
- On peut **assigner une fonction anonyme à une variable**. Cette variable s'utilise alors comme une fonction classique

```
function somme(a, b, c){
  var v1 = (typeof a !== 'undefined')?a:0;
  var v2 = (typeof b !== 'undefined')?b:0;
  var v3 = (typeof c !== 'undefined')?c:0;
  return v1+v2+v3;
}
somme(4) // 4
somme(4,2) // 6
somme(4,2,1) // 7
```

```
function somme(a=0, b=0, c=0){
   return a+b+c;
}
somme(4) // 4
somme(4,2) // 6
somme(4,2,1) // 7
```

```
function (arguments) {
  // Le code de votre fonction anonyme
}
```

```
var sayHello = function() {
    alert('Bonjour !');
};
sayHello();
```

TABLEAUX : CRÉATION, ACCÈS ET MODIFICATION

```
var tab = ['orange', 3, 'melon', 8];
// Equivalent à : var tab = new Array('orange', …);

console.log(tab); // Affiche ["orange", 3, "melon", 8]
console.log(tab[0] + tab[2]); // Affiche "orangemelon"
console.log(tab[1] + tab[3]); // Affiche 11
```

```
var fruits = ["banane", "pomme", "kiwi"];
console.log(fruits.length); // 3
fruits[4] = "mangue";
console.log(fruits.length); // 5 (!!)
console.log(fruits);
// => ["banane", "pomme", "kiwi", empty, "mangue"]
console.log(fruits[3]); // undefined

fruits.sort(); // (Tri alphabétique par défaut)
console.log(fruits);
// => ["banane", "kiwi", "mangue", "pomme", empty]
```

TABLEAUX : QUELQUES AUTRES FONCTIONS

```
// Tableau de nombres (non trié)
var tab = [1, 6, 3, 22, 111];

// Tri "alphabétique" par défaut
tab.sort();
console.log( tab );
// => [1, 111, 22, 3, 6]

// Définition d'un tri "numérique"
tab.sort(function (a, b) {
  if (a < b) {return -1;}
  else if (a > b) {return 1;}
  else {return 0;}
});
console.log( tab );
// => [1, 3, 6, 22, 111]
```

```
var t = [1, 8, 3, 12];
var last = t.pop(); // (r = 12)
var first = t.shift(); // (r = 1)
console.log(t); // [8, 3]
t.push(8);
t.unshift(42);
console.log(t); // [42, 8, 3, 8]

console.log( t.indexOf(8) );
// => 1 (Car 1ère occurence)
/* Mais il y a aussi : lastIndexOf,
  join, slice, ... (cf. la doc) */
```

=> Pour plus d'exemples et le détail des méthodes sur les tableaux JS (Array) :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array

TABLEAUX : DIFFÉRENTS PARCOURS POSSIBLES

- On peut parcourir un tableau grâce à **une boucle for "classique" :**

```
for(var i=0; i < tab.length; i++) {
   console.log(tab[i]);
}
// Affiche : 1 3 6 22 111
```

- Mais il existe aussi des **boucles for "spéciales" :**

```
// for...in parcourt les indexes/propriétés
for (var index in tab) {
  console.log(index + "=>" + tab[index]);
}
// Affiche : 0=>1 1=>3 2=>6 3=>22 4=>111
```

```
// for...of parcourt les valeurs
for (var val of tab) {
  console.log(val);
}
// Affiche : 1 3 6 22 111
```

```
tab.forEach(
  maVal => console.log(maVal)
);
// Applique une fct° anonyme
// pour chaque élément
// du tableau
```

LES OBJETS EN JAVASCRIPT (1)

- Pour créer un **objet littéral**, on peut utiliser des **accolades** (contrairement à un tableau qui utilise des **crochets**) :

```
var unTableau = [];
var unObjet = {};
var unAutreObjet = new Object();
```

- On peut **ajouter dynamiquement** des propriétés (=PHP) et accéder aux propriétés **comme avec des tableaux associatifs** (~=PHP) :

```
var maVoiture = { marque:'Dacia', modele:'Sandero' };
// Format JSON (JavaScript Object Notation)
maVoiture.annee = 2012;
console.log( maVoiture.annee ); // 2012
console.log( maVoiture["annee"] ); // 2012 (équivalent à au-dessus)
```

- On peut parcourir un objet "littéral" avec un **for ... in** :

```
for(var prop in maVoiture){
   console.log( maVoiture[prop] );
}
```

LES OBJETS EN JAVASCRIPT (2)

- On peut facilement **ajouter des méthodes à un objet :**

```
// Définition à la création de l'objet
var maVoiture = {
 marque: 'Dacia', modele: 'Sandero',
 getNom: function(){
   return this.marque+" "+this.modele;
 }
};
```

```
// Définition après-coup
maVoiture.getNom = function(){
   return this.marque
     + " " + this.modele;
};
```

```
console.log(maVoiture.getNom());
// Affiche : "Dacia Sandero"
```

- Même si le JavaScript est un langage objet **basé sur des prototypes**, il est possible de **définir des (simili-)classes** (qui sont en fait des "objets prototypiques")
  (1ère partie de l'article **à lire** : MDN - Le modèle objet JavaScript en détails)

LES CONSTRUCTEURS ET LES CLASSES

- Du coup (même si cela peut paraitre étrange au départ), **toute fonction JavaScript peut être utilisée comme constructeur** :

```
// Ceci est un constructeur
// (une ancienne écriture qui reste toujours valable)
function Person(nick, age, sex, friends){
   this.nick = nick;
   this.age = age;
   this.sex = sex;
   this.friends = friends;
}
```

- Et on utilise le mot-clé "**new**" pour créer de nouvelle instances :

```
var seb = new Person('Sébastien', 23, 'm', []);
var lau = new Person('Laurence', 19, 'f', []);

console.log(seb.nick); // "Sébastien"
console.log(lau.nick); // "Laurence"
```

MANIPULATION D'OBJETS

- On peut **accéder** et **manipuler** librement les propriétés d'un objet :

```
// Modification de l'âge de Sébastien
seb.age = 24;
// Ajout d'un ami à Sébastien
seb.friends.push(new Person('Johann', 19, 'm', []));
```

- Pour manipuler des **dates** en JavaScript, il existe la "classe" **Date** :

```
var past = new Date('2013-12-04 10:00:00');
var now = new Date();
console.log(past.getUTCFullYear()); // 2013
console.log(now.getUTCFullYear()); // 2019
```

Utilisation du prototype pour définir une méthode à une classe

- En JS, le **prototype d'un objet** est utilisé pour fournir de façon dynamique des propriétés aux objets qui héritent du prototype (Plus de détails ici : MDN - Object.prototype)

```
function Person(nick, age, sex, friends){
  this.nick = nick; this.age = age;
  this.sex = sex; this.friends = friends;
}
var seb = new Person('Sébastien', 23, 'm', []);

// Passage par la propriété spéciale "prototype" pour définir
// la méthode addFriend() à tous le objets de Person
Person.prototype.addFriend = function(nick, age, sex, friends){
  this.friends.push(new Person(nick, age, sex, friends));
};

seb.addFriend('Johann' , 19, 'm', []);
```

- Grâce à cela, on peut faire **un système d'héritage en JS**, mais nous n'en parlerons pas dans ce cours (Cf. MDN - Le modèle objet JavaScript en détails)

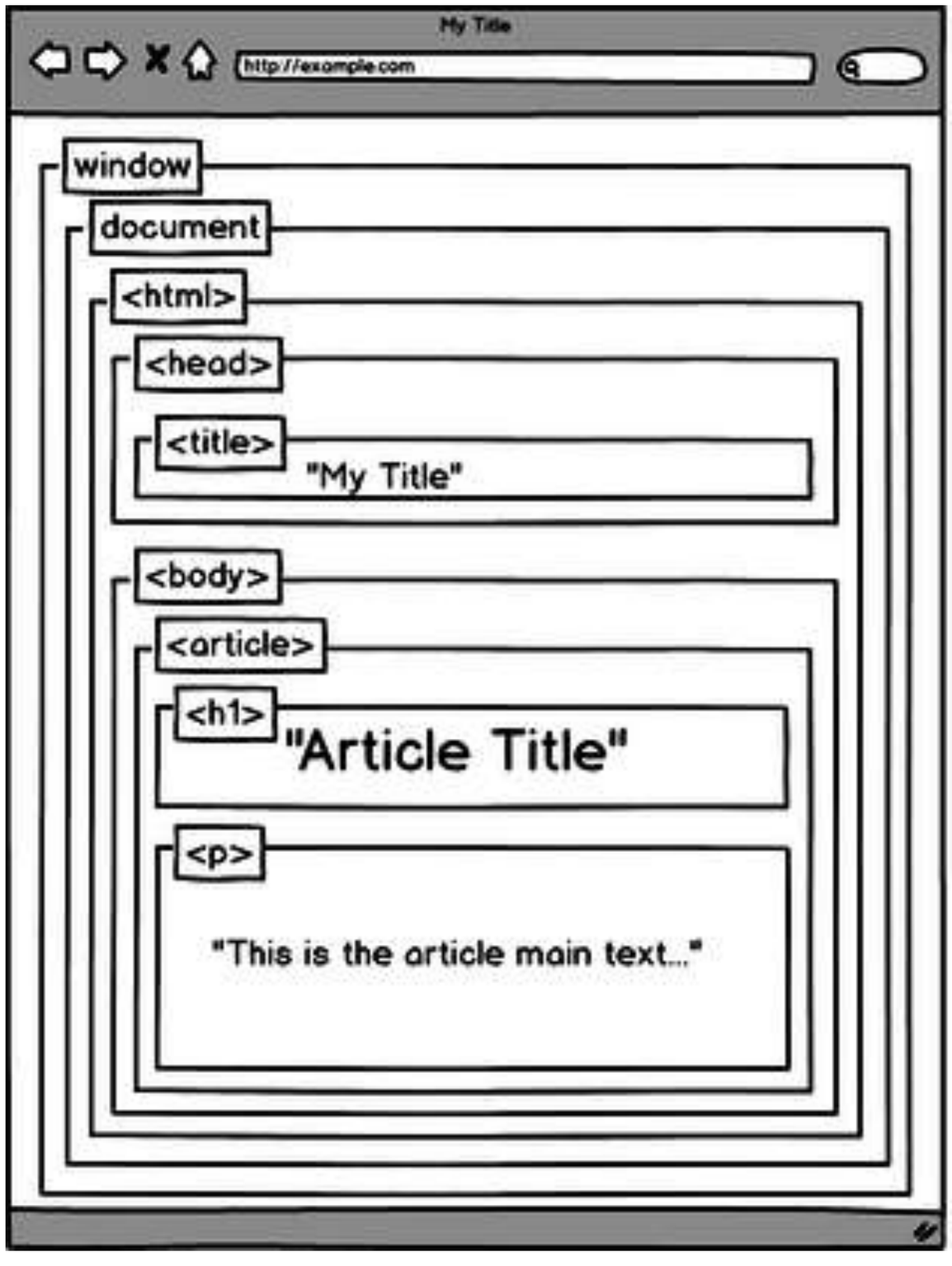
Une nouvelle façon de définir des classes (ECMAScript 2015)

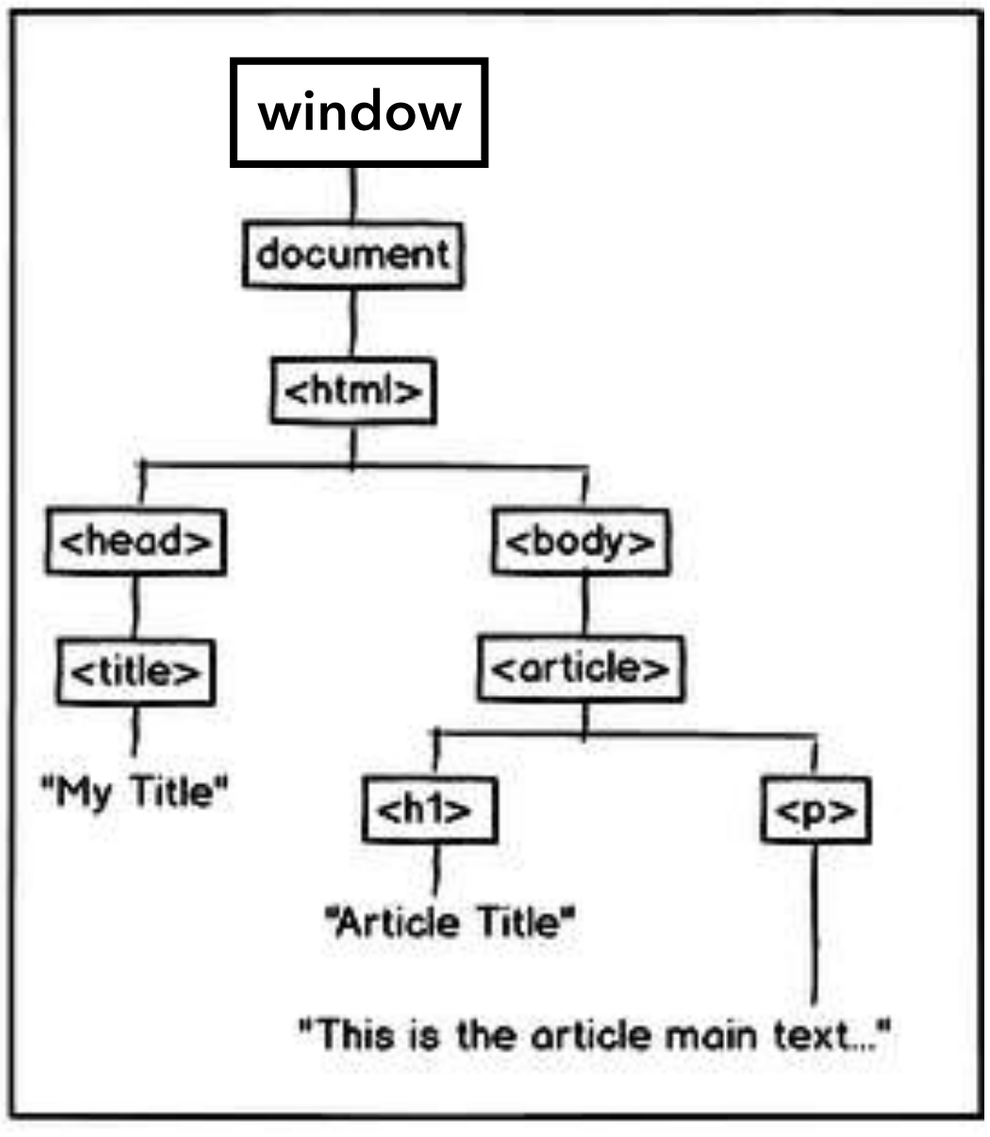
- Depuis ECMAScript 2015, il existe **une syntaxe plus simple** pour créer des objets et manipuler l'héritage en JS (mais rien ne change sur le fond, c'est simplement un "sucre syntaxique" !)

```
class Person {
  constructor(n, a, s, f){
    this.nick = n; this.age = a; this.sex = s; this.friends = f;
  }
  addFriend(nick, age, sex, friends){
    this.friends.push(new Person(nick, age, sex, friends));
  }
  //(La méthode addFriend() reste rattaché au prototype !)
}

class Student extends Person {
  //(Ça reste de l'héritage prototypal !)
  constructor(nick, age, sex, friends, modules){
    super(nick, age, sex, friends);
    this.modules = modules;
  }
}
```

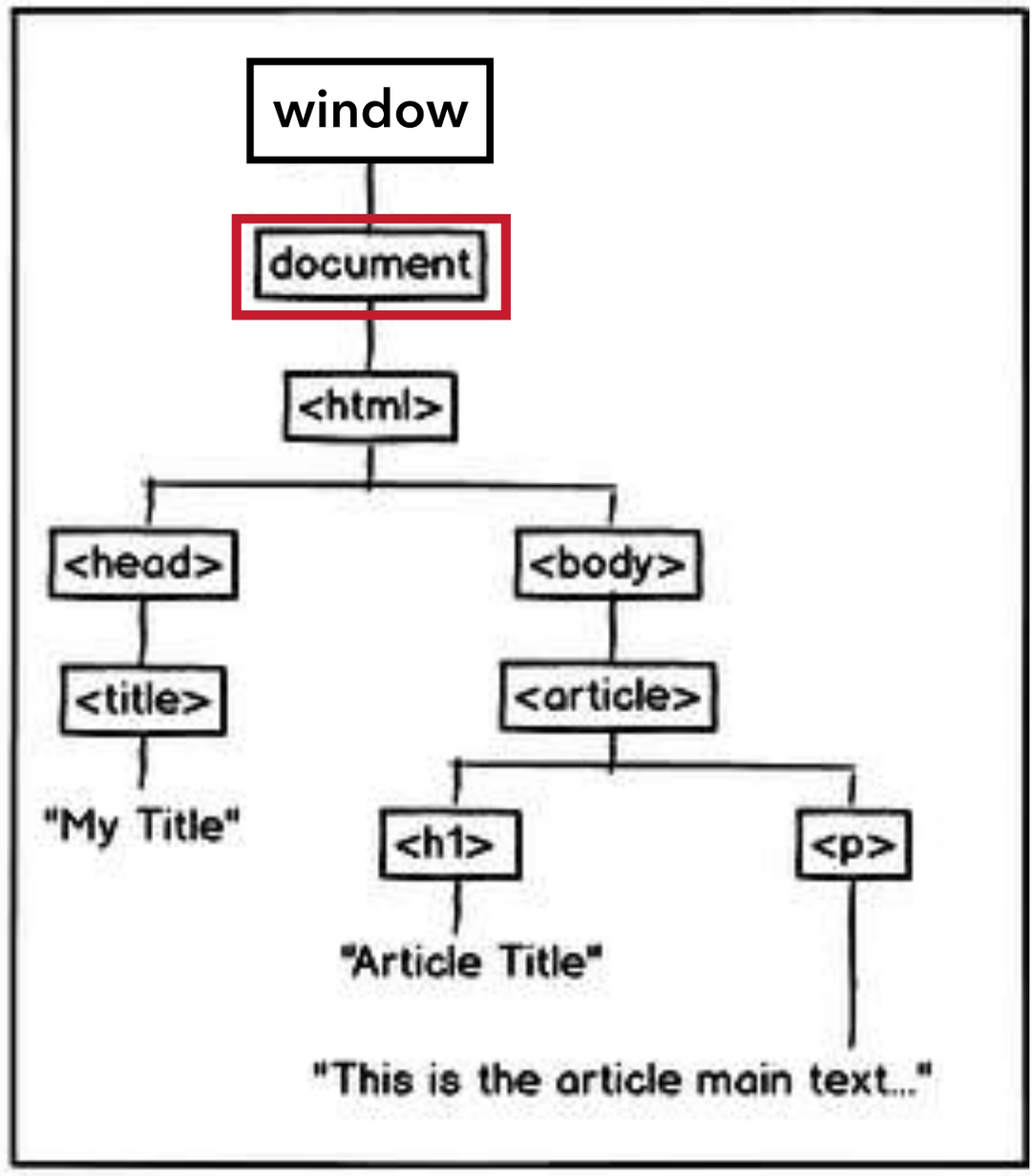
LE DOM : LA BASE DE L'INTERACTION ENTRE L'HTML ET LE JS (1)





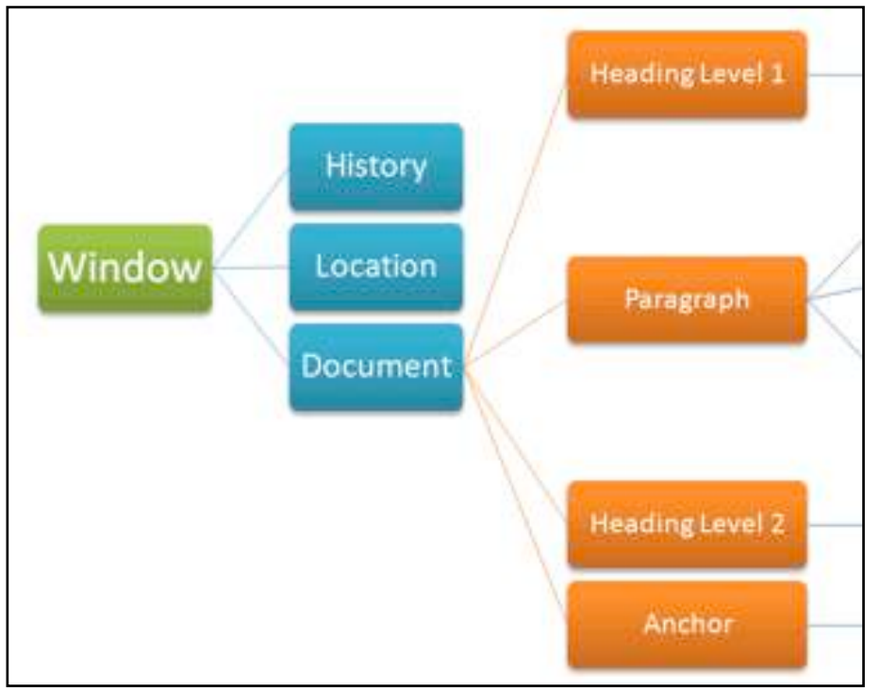
LE DOM : LA BASE DE L'INTERACTION ENTRE L'HTML ET LE JS (2)

- Le **Document Object Model** (abrégé **DOM**) est une **interface de programmation standardisé** pour les documents **XML** et **HTML**
- Le **DOM** prend la forme d'un **objet** (ayant comme nom "document") ayant des **propriétés** et des **méthodes** permettant d'accéder à la page web (d'où le nom d'interface/ API)
- Chaque élément est appelé **noeud** ("**node**" en anglais)



L'OBJET GLOBAL "WINDOW"

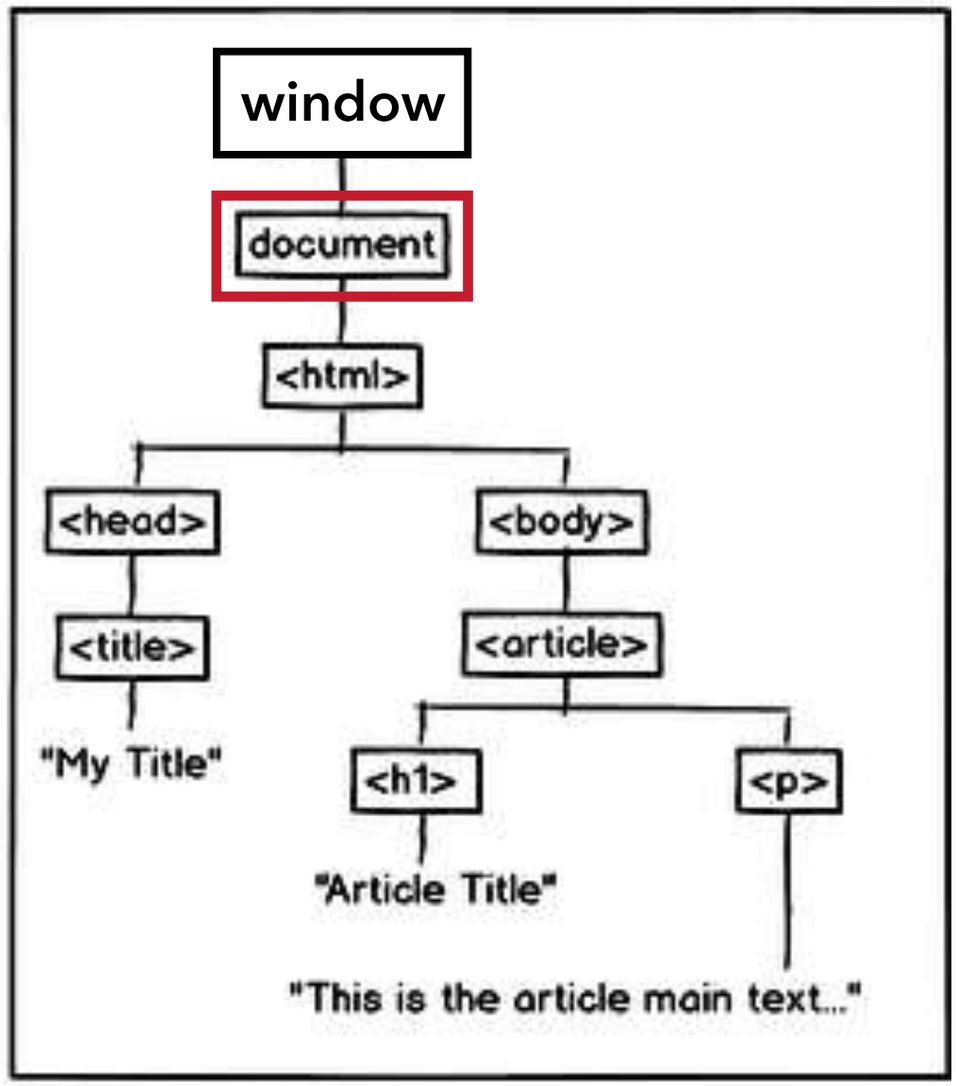
- L'objet **window** (pré-défini & géré par tous les navigateurs) est ce qu'on appelle **un objet global** qui **représente la fenêtre du navigateur** (Par exemple : "window.location" permet d'accéder à l'URL actuelle)
- C'est **l'objet principal** dans le modèle objet javascript, il est le **parent de chaque objet** qui compose la page web
- *Pour information, les fonctions **alert(), prompt(), confirm()** ne sont en fait pas vraiment des fonctions globales mais **des méthodes de l'objet window***




```
window.alert("Avec window");
alert("Sans window");
```

L'OBJET "DOCUMENT"

- L'objet **document** est un sous-objet de window (un des ses "enfants") et il représente **la page Web**, et plus précisément **l'ensemble du contenu de la balise <html>**
- C'est grâce à cet élément-là que nous allons **pouvoir accéder aux éléments HTML et les modifier** (via le DOM)

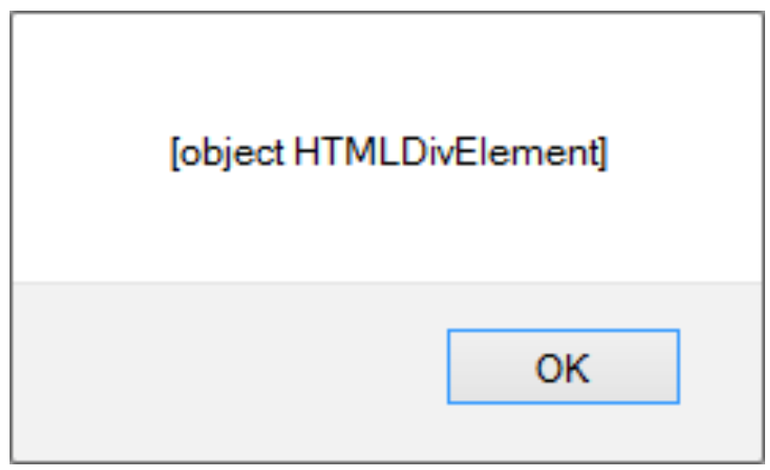



```
<body>
   Le contenu de ma page
   <script>
      var texte = document.body.textContent;
      console.log(texte); // "Le contenu de ma page"
      document.body.textContent = "Le nouveau texte";
      // La page affiche alors "Le nouveau texte"
   </script>
</body>
```

ACCÈS AUX ÉLÉMENTS HTML

- L'objet **document** a plus de **50 propriétés et méthodes** (même s'il faut faire attention car une partie est maintenant obsolète !)
- Pour **accéder aux éléments HTML**, les principales méthodes sont :
  - ➡ **getElementById()** qui permet d'accéder à un élément en connaissant son **ID** (qui est simplement l'attribut **id** de l'élément)
  - ➡ **getElementsByTagName()** qui permet de récupérer, sous la forme d'un tableau, tous les éléments de la "famille" indiquée en paramètre (indiquer "**div**" pour récupérer tous les ** ** de la page)

```
<div id="myDiv">
    <p>Un peu de texte <a>et un lien</a></p>
</div>
<script>
   var div = document.getElementById('myDiv');
   alert(div);
</script>
```



ÉDITER LES ÉLÉMENTS HTML : LES ATTRIBUTS

- Modifier les **attributs** d'un élément HTML avec **get...** et **setAttribute()**
  OU directement par **l'attribut accessible via une propriété**

```
<body>
  <a id="myLink" href="http://www.azerty.com">Le lien à modifier</a>

  <script>
    var link = document.getElementById('myLink');
    // Modification de l'attribut href via get/setAttribute()
    var href = link.getAttribute('href');
    console.log(href);
    link.setAttribute('href', 'http://www.google.com');
    // Modification de l'attribut href via la propriété "href"
    href = link.href;
    console.log(href);
    link.href = 'http://www.qwant.com';
  </script>
</body>
```

ÉDITER LES ÉLÉMENTS HTML : LE CONTENU

- Pour modifier le contenu d'un élément HTML, on utilise principalement les propriétés **innerHTML**, **textContent** et **innerText** (cette dernière n'ayant été standardisée qu'en 2016)

```
<body>
  <div id="myDiv">
    <p>Un peu de texte <a>et un lien</a></p> </div>

  <script>
    var div = document.getElementById('myDiv');
    console.log(div.innerHTML);
    // Affiche : "    <p>Un peu de texte <a>et un lien</a></p>\n  "
    console.log(div.textContent);
    // Affiche : "    Un peu de texte et un lien\n  "
    console.log(div.innerText);
    // Affiche : "Un peu de texte et un lien"

    div.innerHTML += " et <strong>une partie en emphase</strong>.";
  </script>
</body>
```

- Il existe d'autres méthodes comme : **document.write()** *(cf. TP1)*

EXEMPLE DE RÉCUPÉRATION ET MODIFICATION VIA LE DOM

```
<!DOCTYPE html>
<html>
 <body>
  <p>Hello World !</p>
  <div id="main">
    <p>Lorem ipsum</p> <p>dolor sit amet</p>
  </div>
  <hr>
  <p id="demo" style="color:red;"></p>

  <script>
   var mainElmt = document.getElementById("main");
   var pElmts = mainElmt.getElementsByTagName("p");
   document.getElementById("demo").innerHTML =
    "Le 1er paragraphe du 'main' est : " + pElmts[0].innerHTML;
  </script>
 </body>
</html>
```

Hello World !

Lorem ipsum

dolor sit amet

Le 1er paragraphe du 'main' est : Lorem ipsum

=> Qu'est-ce que cela affiche selon vous ?

ASSOCIER UNE ACTION À UN ÉVÈNEMENT DANS LE HTML : UN EXEMPLE

```
<span onclick="alert('Salut !');" style="color: red;">
    Cliquez ici que je vous salue !</span>
```

Cliquez ici que je vous salue !

- Pour **associer une action simple à un événement** sur un élément HTML, on peut ajouter l'attribut HTML "**onclick**"
- La valeur de cet attribut doit être du **code JavaScript** (une ou plusieurs instruction(s)) entre guillemets ("...")
- Il existe d'autre attributs HTML en lien avec des évènements : **ondblclick**, **onmouseover**, ...

ASSOCIER UNE ACTION UN ÉVÈNEMENT EN JS (2 MÉTHODES)

```
function myFunc(){
  alert( Date() );
}

// Méthode par ajout d'un attribut "onclick" (DOM-0)
document.getElementById("myBtn").setAttribute("onclick", "myFunc()");
// OU
document.getElementById("myBtn").onclick = myFunc;

// OU Méthode par utilisation de addEventListener() (DOM-2)
// (qui permet, entre autres, d'associer plusieurs fonctions à un
évènement)
document.getElementById("myBtn").addEventListener("click", myFunc);
```

Quelques autres évènements

- Voici **la liste des principaux événements existants**, ainsi que les actions à effectuer pour qu'ils se déclenchent :

Nom de l'événement	Action pour le déclencher
`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
`focus`	« Cibler » l'élément
`blur`	Annuler le « ciblage » de l'élément