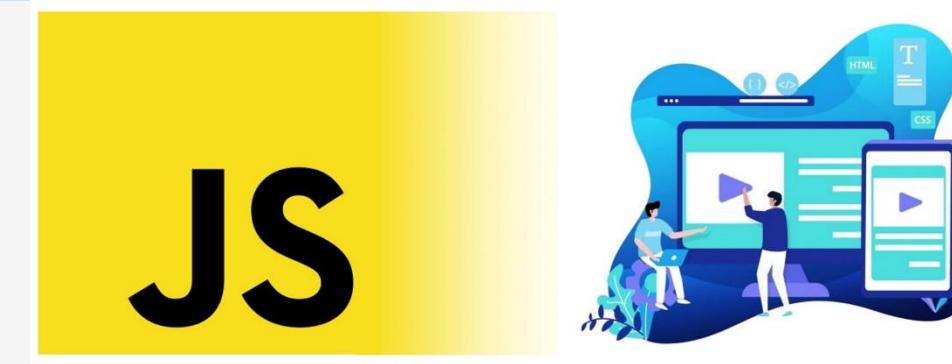


Formation :

JavaScript et HTML dynamique (3 jours)

Fondements et pratiques



Formateur

Dr. Bassem Seddik
Maître Technologue en informatique
Consultant en IT, entrepreneur
M2ii - INETUM - JavaScript et HTML dynamique
Bassem.seddik@gmail.com

Objectifs pédagogiques de la formation

- Maîtriser la syntaxe du langage JavaScript
- Manipuler la structure DOM d'une page HTML
- Gérer la programmation événementielle
- Interagir avec les feuilles de style CSS
- Gérer des échanges asynchrones AJAX

Travaux pratiques

- Programmation JavaScript, Manipulation du DOM, Gestion de l'interactivité, etc.

Evaluation des prérequis sur: <https://www.orsys.fr/eval/test/DHL>

Programme de formation

I- Les technologies du Web

II- Le langage JavaScript, prise en mains

III- Evénements et données

IV- Gestion de formulaires HTML

V- Interaction avec les CSS

VI- Manipulation du DOM XML

VII- Ajax

Présentation du formateur

Bassem Seddik:

- Docteur en informatique,
- 15 ans d'expérience professionnelle
 - 10 ans comme enseignant universitaire
 - Spécialisé en Développement Web et Multimédia 2D/3D
- Formateur et consultant IT



I- Les technologies du Web

1. JavaScript: Historique et applications
2. Versions de JS et impact sur la portabilité des programmes, ECMA,
3. JavaScript dans un navigateur Web et ses composants: HTML, XHTML, CSS, JS,...
4. Types de navigateurs et compatibilité inter Navigateurs
5. Les relations inter-technologies: Web et positionnement de JS
6. Web 2.0 et importance de JS pour le Web 2.0
7. Echanges de données, protocoles et utilisation dynamique de JS
8. Les outils de développement (éditeur, débogueur...).

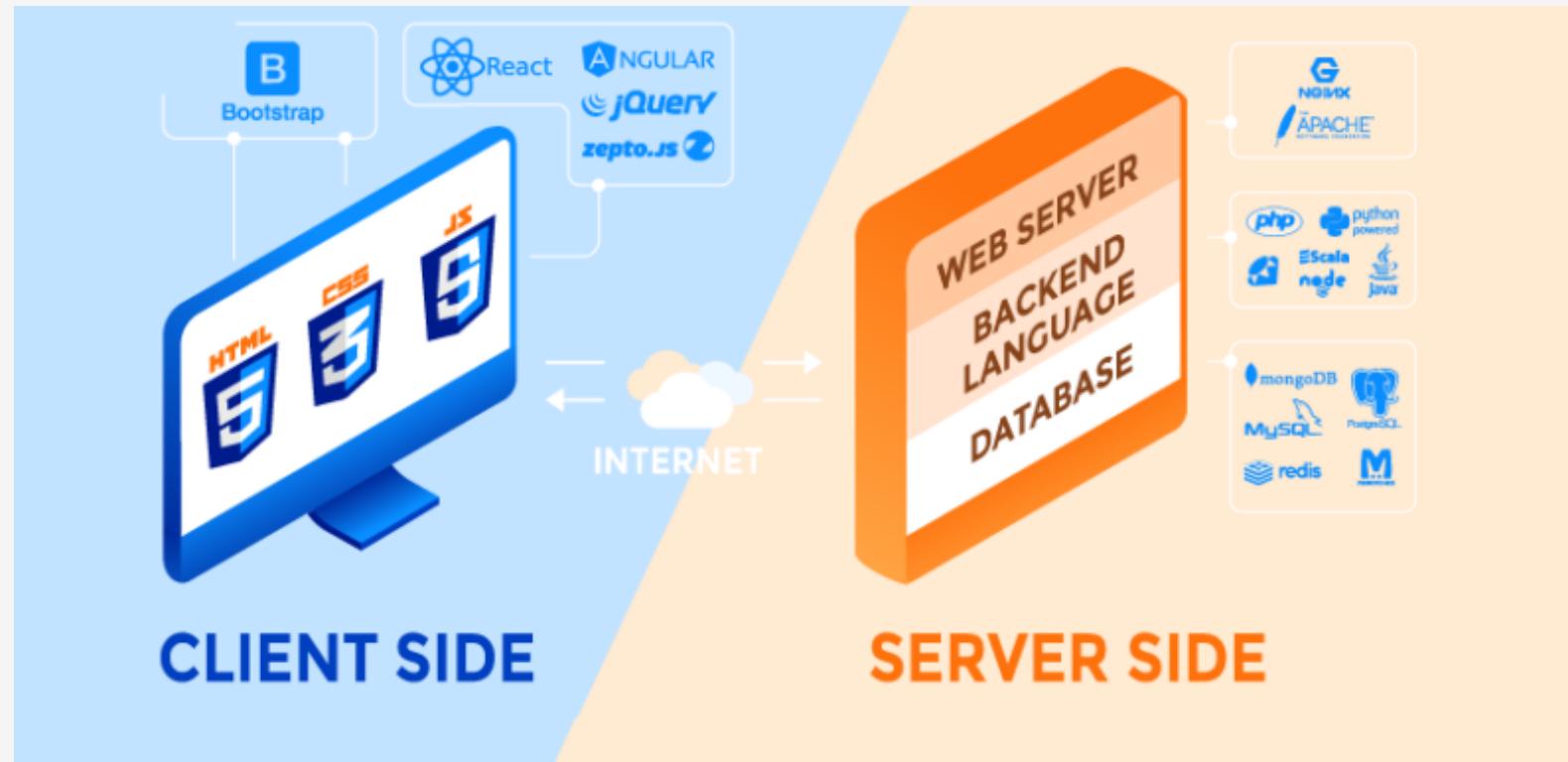
JavaScript: Historique et applications

- JavaScript est un langage créé **pour le Web**
- Si vous êtes familier avec l'utilisation de **HTML** et **CSS** pour créer des pages Web, JavaScript est la dernière pièce dont vous aurez besoin pour **donner vie** à vos sites Web.
- En 1995, lorsque *Brandan Eich* a créé un JavaScript, il l'a fait pour faciliter l'ajout d'éléments **interactifs et dynamiques** aux sites Web.
 - D'ailleurs, il a participé à la création du navigateur Mozilla
 - Vous [pouvez vérifier ici](#), Mozilla offre une des meilleures documentations sur JavaScript
- Aujourd'hui, JavaScript est utilisé pour toutes sortes d'applications.
 - De la programmation d'un **robot** avec notre **Arduino**,
 - à l'écriture d'un jeu avec un script sous le **moteur de jeux Unity**,
 - Des graphiques 2D/3D impressionnantes (**Three.js**),
 - Même de l'intelligence artificielle (**TensorFlow.js**),
 - Même certains **éditeurs de code (Visual Studio Code)** ont été construits avec JavaScript (**TypeScript**), ...
- Tout simplement, les opportunités ouvertes maintenant avec JavaScript sont infinies.



Client VS Serveur

- Tout comme le HTML, le JavaScript est généralement exécuté par le navigateur de l'internaute : on parle d'un comportement **client-side**, par opposition au **server-side** lorsque le code est exécuté par le serveur.



Versions de JS et impact sur la portabilité des programmes

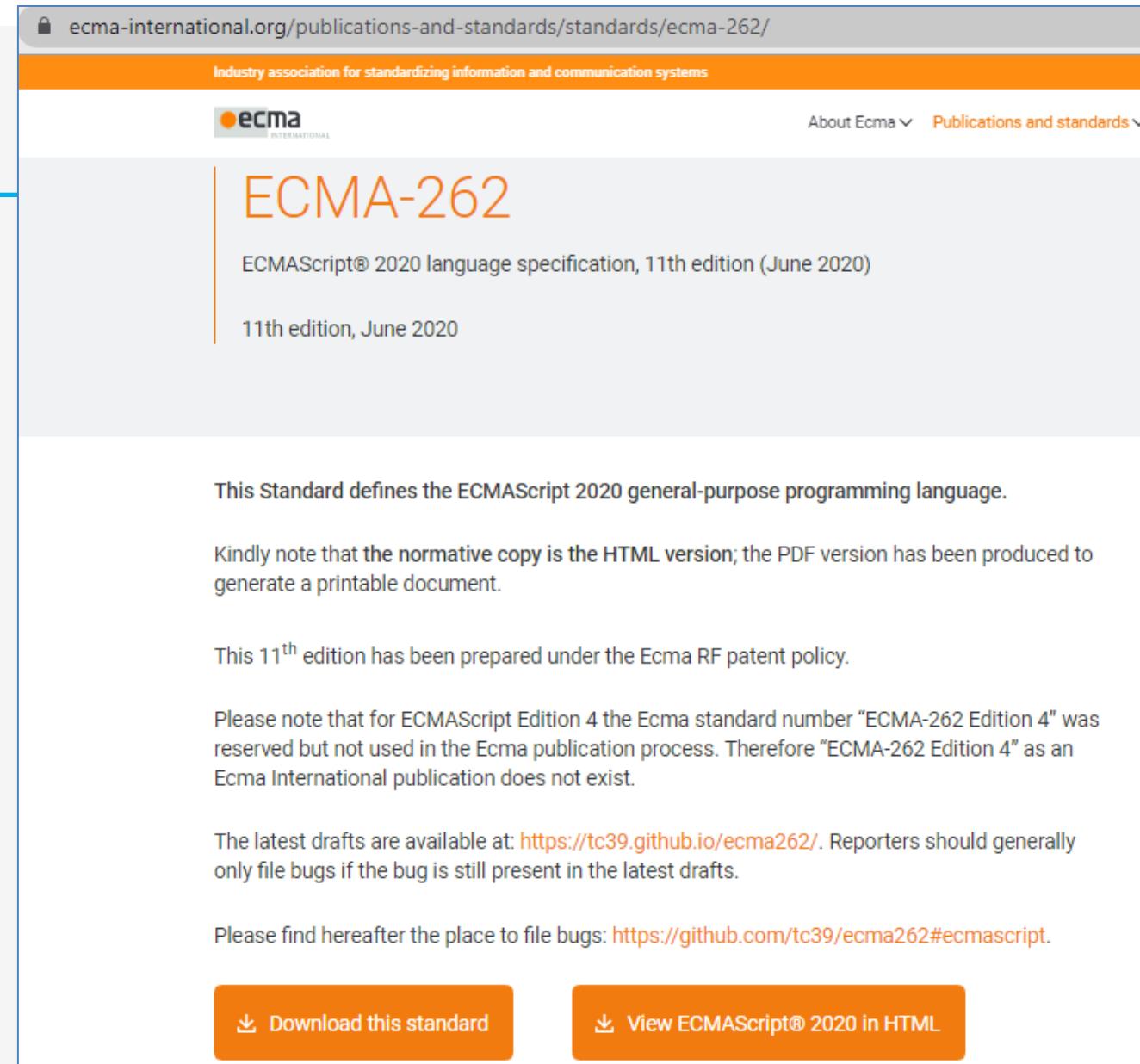
JavaScript est un **Langage interprété** : dans ce cas, il n'y a pas de compilation.

- Le code source reste tel quel, et si on veut exécuter ce code, on doit le fournir à un interpréteur qui se chargera de le lire et de réaliser les actions demandées.
 - Il s'appelait au départ **LiveScript**, mais ses créateurs ont rapidement migré vers le nom JavaScript pour gagner la popularité du label « **Java** »
 - Mais attention! Ils sont totalement différents! D'ailleurs, Java lui, est un langage compilé
 - Aussi Microsoft a créé après un autre langage: **JScript**! Lui, il est était semblable à JS!
- Logiquement, après un bout de temps, JavaScript a été standardisé par l'ECMA International sous le nom d'**ECMAScript** qui constitue la nouvelle référence.
- La dernière version standardisée du JavaScript est basée sur l'ECMAScript 5, sorti en 2009. puis, ECMAScript 6.
 - Maintenant, on fait référence par année tel que ES2016, ES2017, etc.



ECMA: l'organisation

- Ecma (**European Computer Manufacturers Association**) International est une association de l'industrie dédiée à la normalisation des systèmes d'information et de communication
- ECMAScript/JavaScript sous sa dernière édition de Juin 2020 est baptisé [ECMA-262](#)
- D'une manière générale, JavaScript désigne les dialectes du langage ECMAScript



The screenshot shows the official page for ECMA-262 on the ECMA International website. The page title is "ECMA-262" and it specifies the "ECMAScript® 2020 language specification, 11th edition (June 2020)". It notes that the normative copy is the HTML version, and the PDF version is for generating a printable document. It also mentions the preparation under the Ecma RF patent policy and the availability of latest drafts and bug reports. Two prominent orange buttons at the bottom are "Download this standard" and "View ECMAScript® 2020 in HTML".

ecma-international.org/publications-and-standards/standards/ecma-262/

Industry association for standardizing information and communication systems

ecma INTERNATIONAL

About Ecma ▾ Publications and standards ▾

ECMA-262

ECMAScript® 2020 language specification, 11th edition (June 2020)

11th edition, June 2020

This Standard defines the ECMAScript 2020 general-purpose programming language.

Kindly note that the normative copy is the HTML version; the PDF version has been produced to generate a printable document.

This 11th edition has been prepared under the Ecma RF patent policy.

Please note that for ECMAScript Edition 4 the Ecma standard number "ECMA-262 Edition 4" was reserved but not used in the Ecma publication process. Therefore "ECMA-262 Edition 4" as an Ecma International publication does not exist.

The latest drafts are available at: <https://tc39.github.io/ecma262/>. Reporters should generally only file bugs if the bug is still present in the latest drafts.

Please find hereafter the place to file bugs: <https://github.com/tc39/ecma262#ecmascript>.

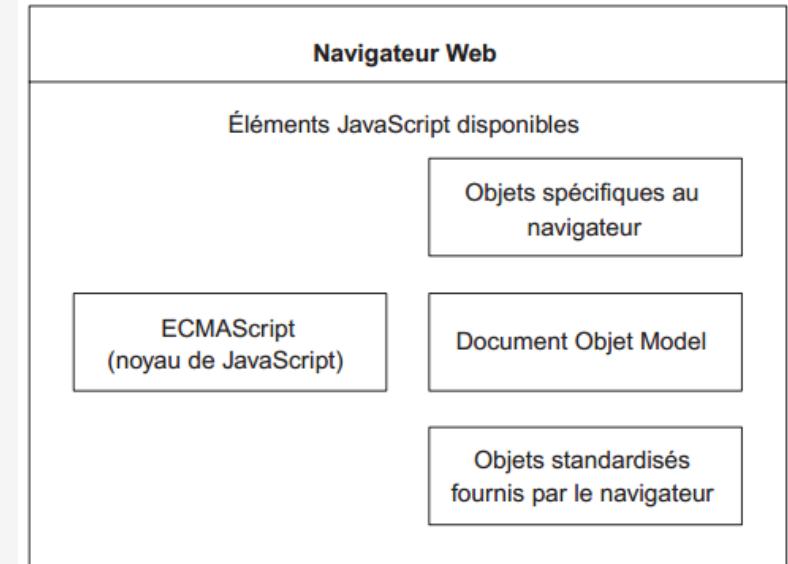
[Download this standard](#)

[View ECMAScript® 2020 in HTML](#)

JavaScript dans un navigateur Web

JavaScript propose différents mécanismes permettant de faire interagir avec le **DOM** et les éléments écrits avec les langages **HTML**, **xHTML** et **CSS** d'une page Web avec les scripts JavaScript.

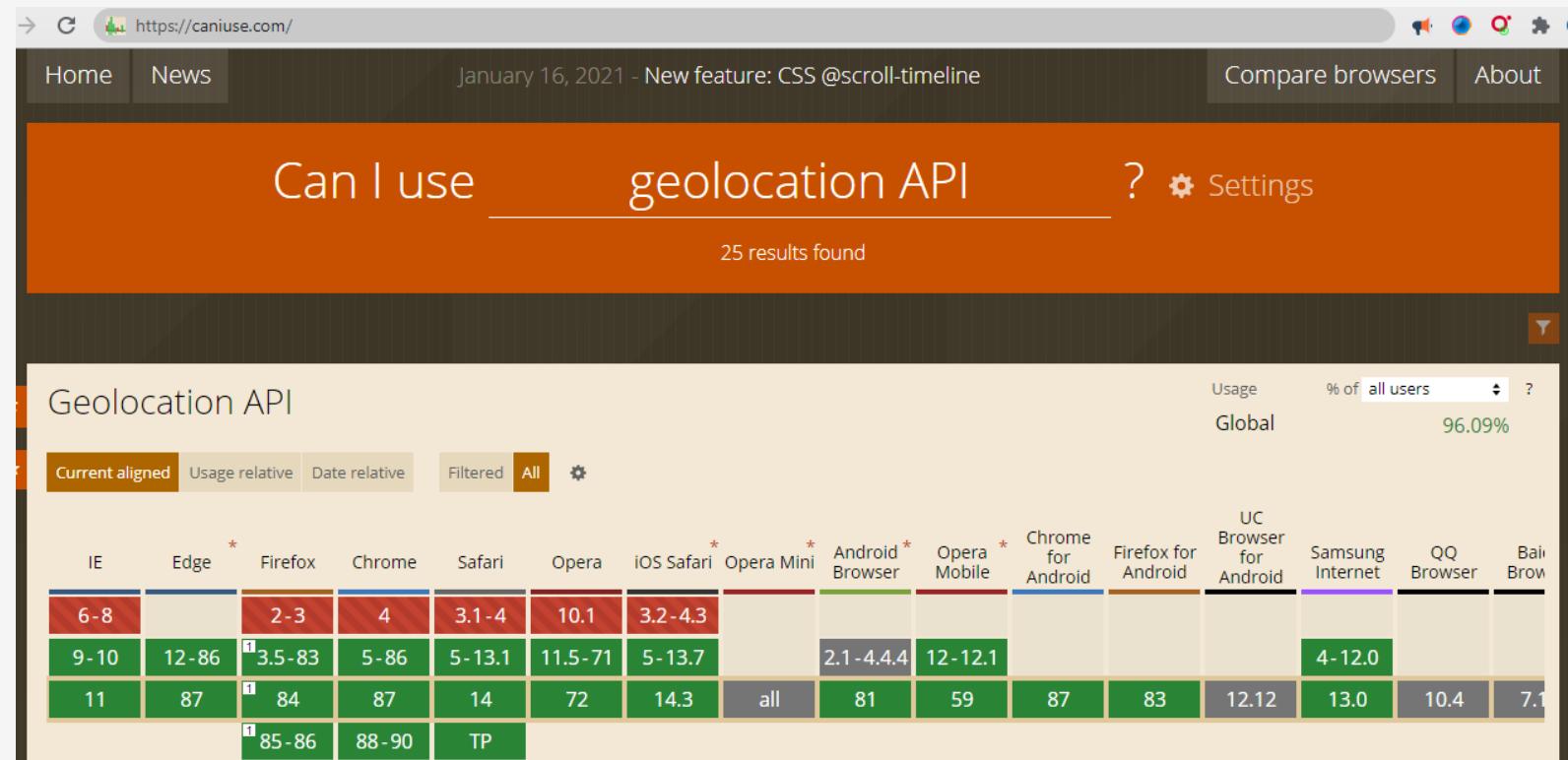
- **DOM** (Document Object Model) est une représentation en mémoire d'un arbre XML normalisée sous forme d'objets.
- **xHTML** (eXtensible HyperText Markup Language): Positionné en tant que successeur du HTML, est un langage distinct, qui reformule le HTML.
 - De ce fait, les pages Web utilisant le xHTML doivent être bien formées au sens XML.
 - Toutes les balises doivent être correctement fermées et les attributs délimités par le caractère « ou '.
- **CCS** (Cascading Style Sheets) adresse les problématiques d'affichage des pages HTML.
 - Son objectif est de permettre une séparation claire entre la **structure** d'une page, par le biais des langages HTML ou xHTML, et sa **présentation**, décrite avec CSS



Types de navigateurs et compatibilité inter Navigateurs,

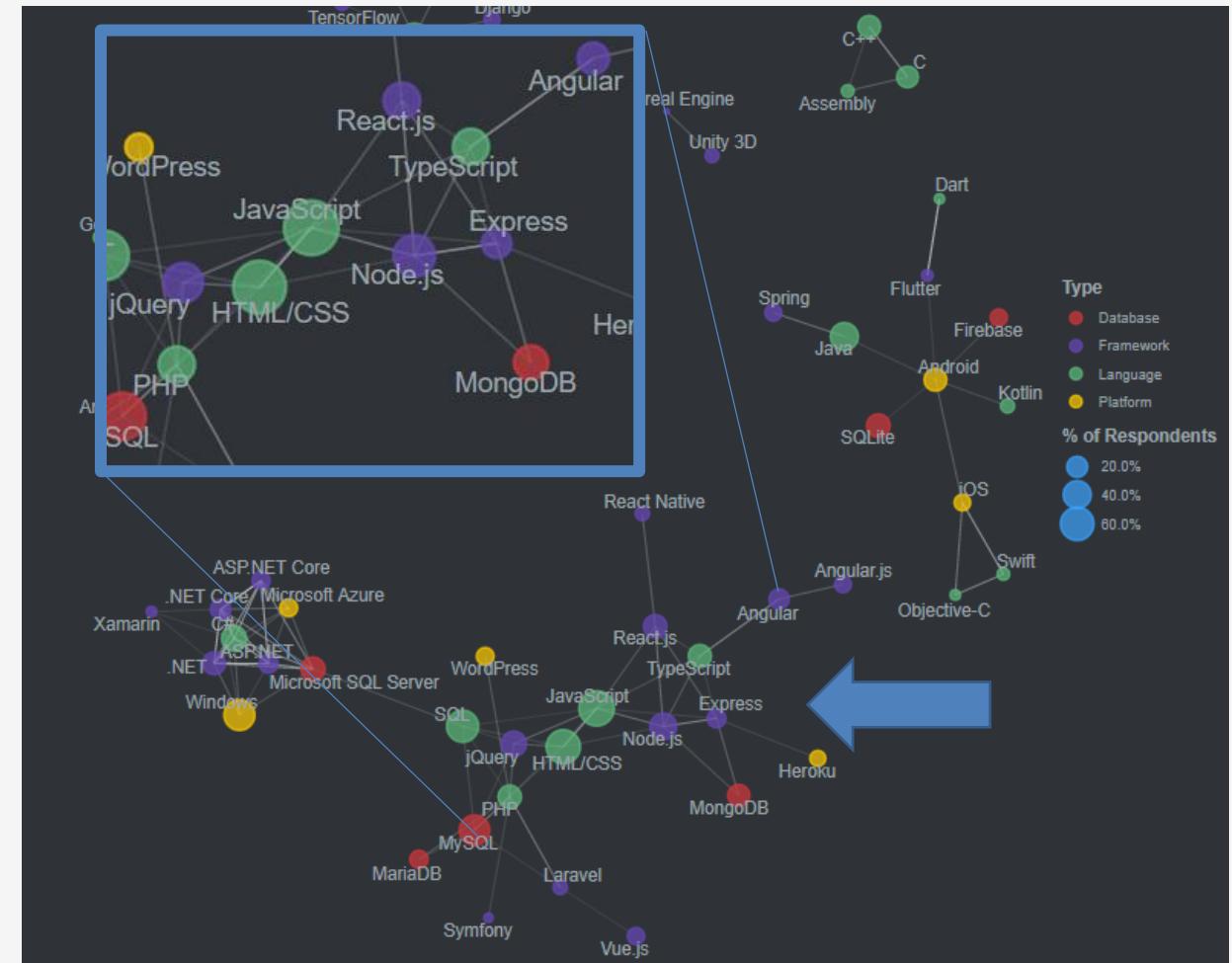
- Le site "**Can I use**" fournit des tables de support de navigateur à jour pour le soutien des technologies Web **front-end** sur les navigateurs Web de bureau et mobiles.

<https://caniuse.com/>



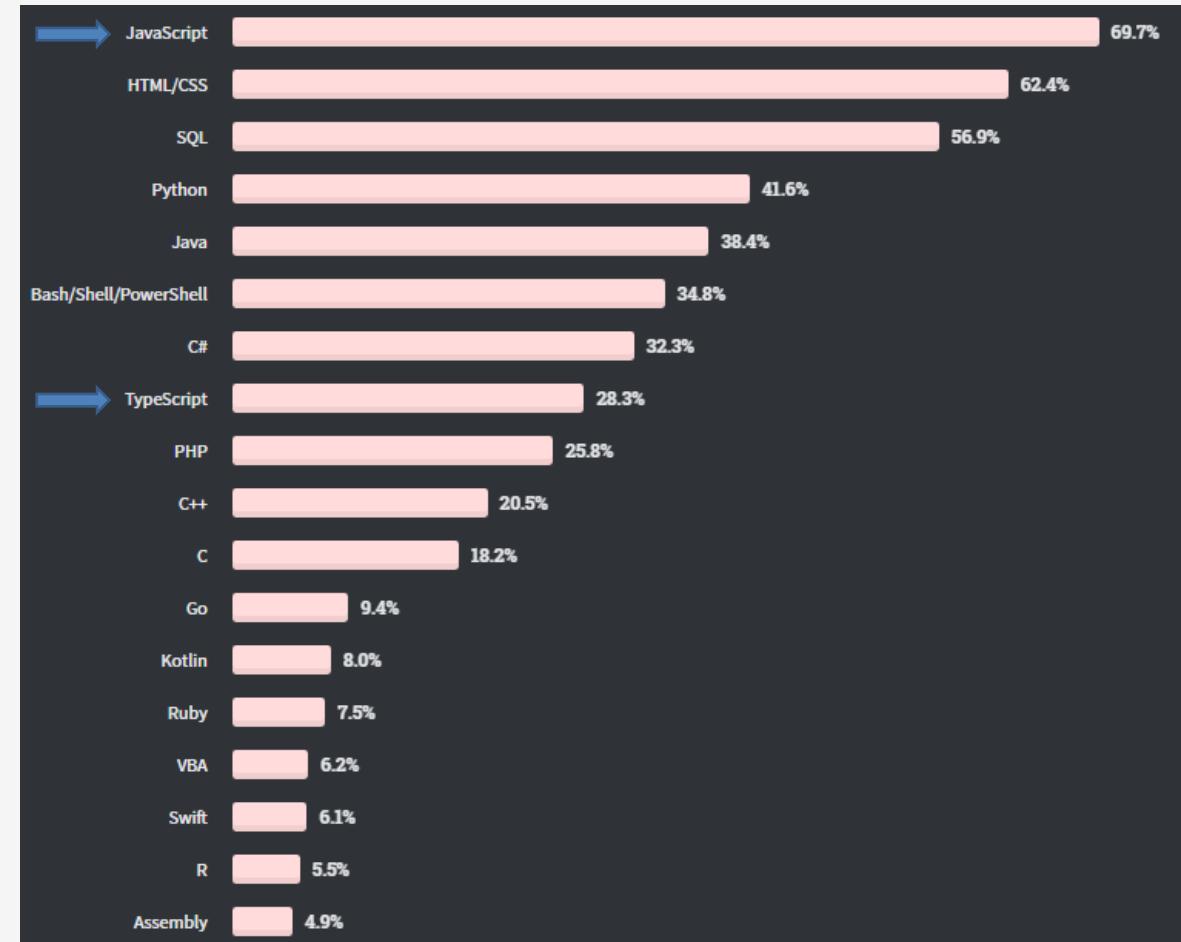
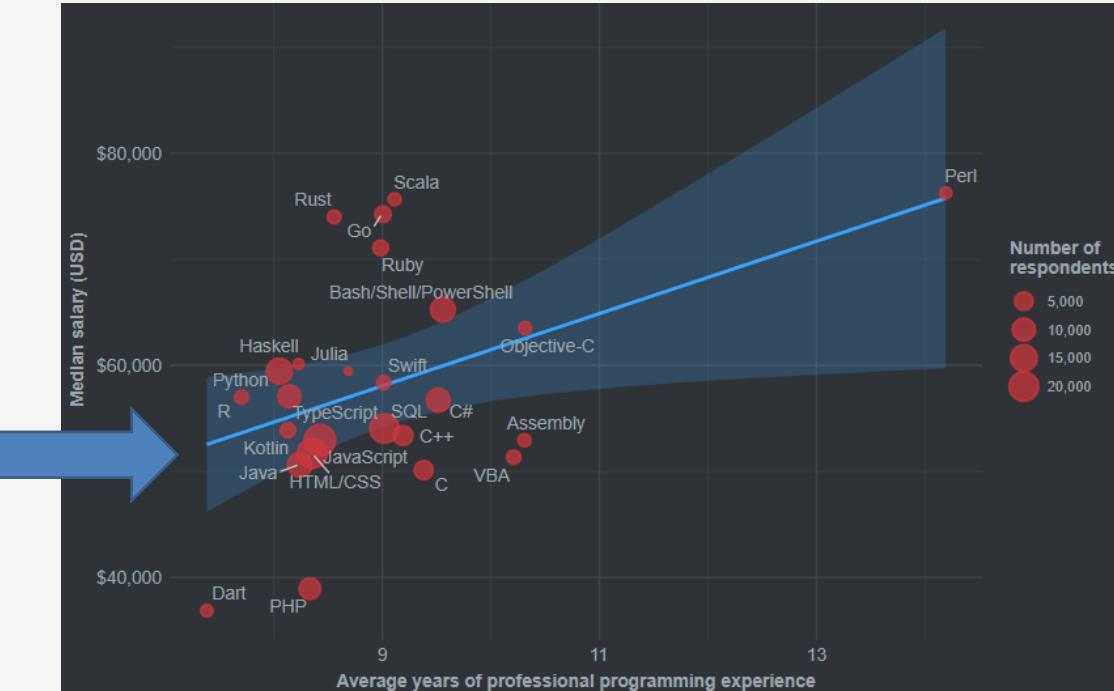
Les relations inter-technologies: Focus Web

- Les technologies se regroupent en écosystèmes connexes qui ont tendance à être utilisés par les mêmes développeurs.
- Ce graphique en nœuds le démontre en montrant quelles technologies sont les plus corrélées les unes avec les autres.
- Noter la **position centrale** de **JavaScript** et de sa version récente **TypeScript** en plein milieu de l'écosystème Web
- Référence: Les technologies les plus populaires selon le Survey de **STACKOVERFLOW** de toute l'année 2020 passée



Positionnement de JS

- Référence: Les technologies les plus populaires selon le Survey de **STACKOVERFLOW** de toute l'année 2020
- **Rémunérations**: langage de programmation et expérience



Web2.0 et importance de JS pour le Web 2.0

- Le **Web 2.0**, quelquefois appelé **Web participatif**, désigne l'ensemble des techniques, des fonctionnalités et usages qui ont suivi la forme originelle du Web, caractérisé par plus de simplicité et d'interactivité.



Les internautes peuvent :

- d'une part contribuer à l'échange d'informations et interagir (**Web Participatif**) de façon simple, à la fois au niveau du **contenu** et de la **structure** des pages,
 - d'autre part entre eux, créant notamment le **Web social**. Grâce aux outils mis à leur disposition, les internautes construisent le Web. (Wikipedia)

Principles

- Permettre l'interaction et la collaboration entre les utilisateurs
 - Contenu généré dynamiquement et mis à jour fréquemment
 - **JavaScript constitue la clé de voûte du Web 2.0**
 - Nécessité d'avoir un contenu bien formé et valide : XHTML

Echanges de données, protocoles et utilisation dynamique de JS

- JavaScript permet de réaliser des requêtes HTTP en recourant aux mécanismes prédéfinis
- Les pages Web ont désormais la possibilité d'échanger des données avec des applications distantes par l'intermédiaire du protocole HTTP (ou HTTPS) sans avoir à recharger complètement leurs interfaces graphiques.

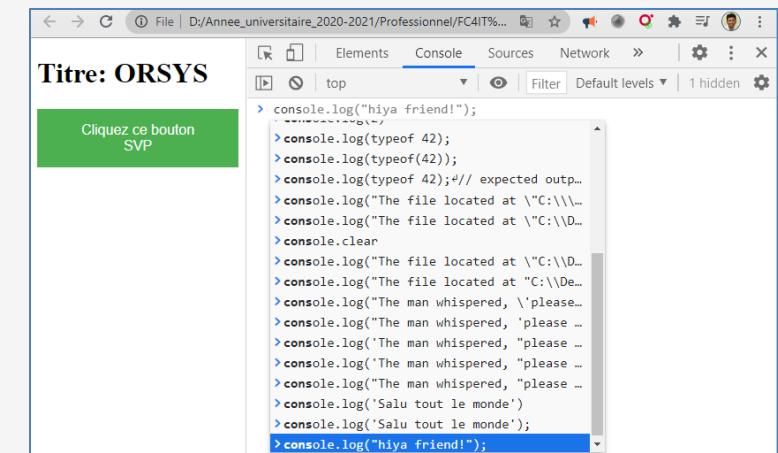
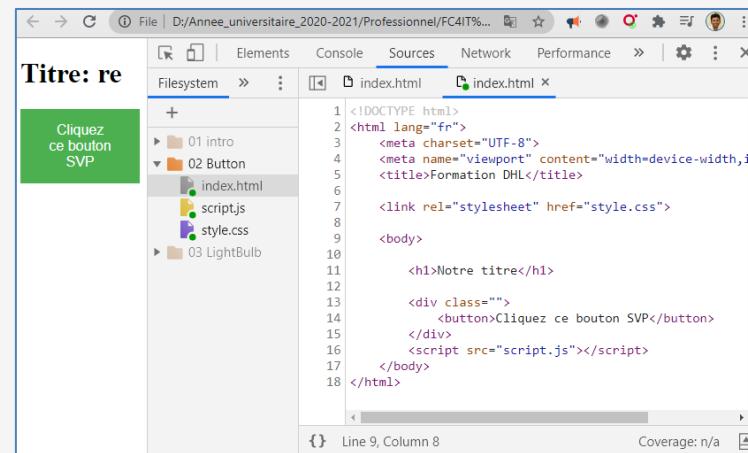
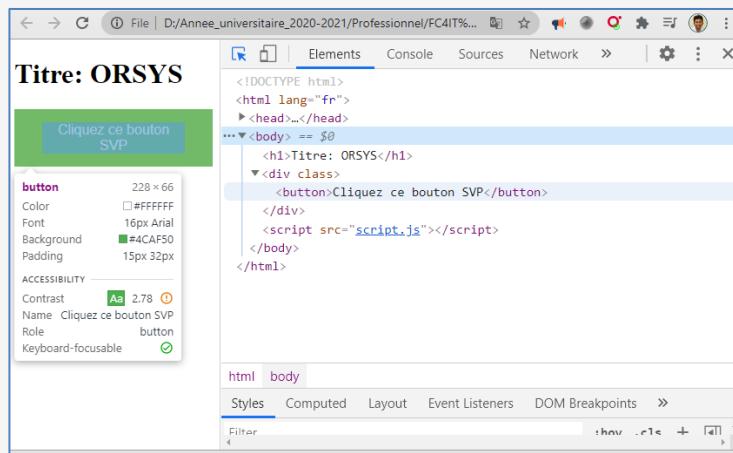
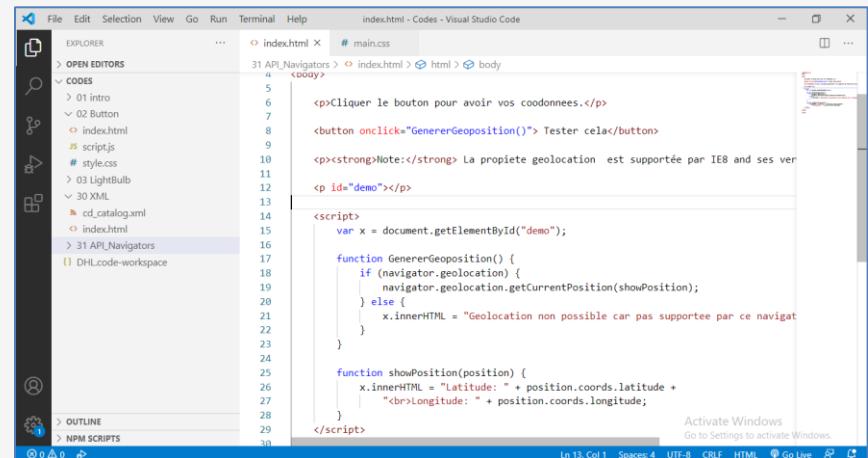
HTTP (HyperText Transfer Protocol) correspond au protocole de transport applicatif de base d'Internet. Il s'agit d'un protocole ASCII transportant des informations sous forme de texte et qui peut fonctionner sur n'importe quel protocole réseau fiable, le plus souvent TCP. HTTPS en est la version sécurisée via cryptage SSL.

- Les techniques d'échange de données sont communément désignées par l'acronyme AJAX (Asynchronous JavaScript and XML). Elles se fondent sur JavaScript afin d'exécuter une requête HTTP dans une page Web et de traiter sa réponse de manière **synchrone** ou **asynchrone**.
- L'une des approches possibles se base sur la classe JavaScript **XMLHttpRequest**, présente dans la plupart des navigateurs récents et en cours de standardisation par le W3C. Cette classe offre la possibilité de 1) exécuter une requête HTTP puis 2) de gérer la réponse.

W3C (World-Wide Web consortium) est un consortium dont l'objectif est de promouvoir les technologies Web (HTML, XHTML, XML, CSS, etc.). Cet organisme n'émet pas de norme mais des recommandations représentant des standards.

Les outils de développement (éditeur, débogueur)

- Mode **inspection** de code dans le navigateur
- Accélération du développement et assistance à la saisie du code
 - Editeur **Visual Studio Code** (écrit en TypeScript), ...
 - Editeur interne pour développeurs sous Chrome ([Sources](#))
 - Evaluation directe via la console du navigateur ([Console](#))
 - [NotePad++](#) : Certainement l'éditeur de texte le plus connu



DEMO

Ce qu'on peut faire avec JavaScript

Voir codes sur [/INETUM JavaScript \(Ceque_JS_fait\)](#)

```
<!DOCTYPE html>
<html lang="fr">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <title>Exemple 01</title>
    <link rel="stylesheet" href="style.css">

    <script src="externalScript.js"></script>

    <body>
        <div class="">
            <h1>Notre titre</h1>
        </div>

        <script>
            var mavaribale = 3;
            document.write( 'Ma nouvelle valeur est : ', typeof (mavaribale));
        </script>
    </body>
</html>
```

Programme de formation

I- Les technologies du Web

II- Le langage JavaScript, prise en mains

III- Evénements et données

IV- Gestion de formulaires HTML

V- Interaction avec les CSS

VI- Manipulation du DOM XML

VII- Ajax

II- Le langage JavaScript, prise en mains

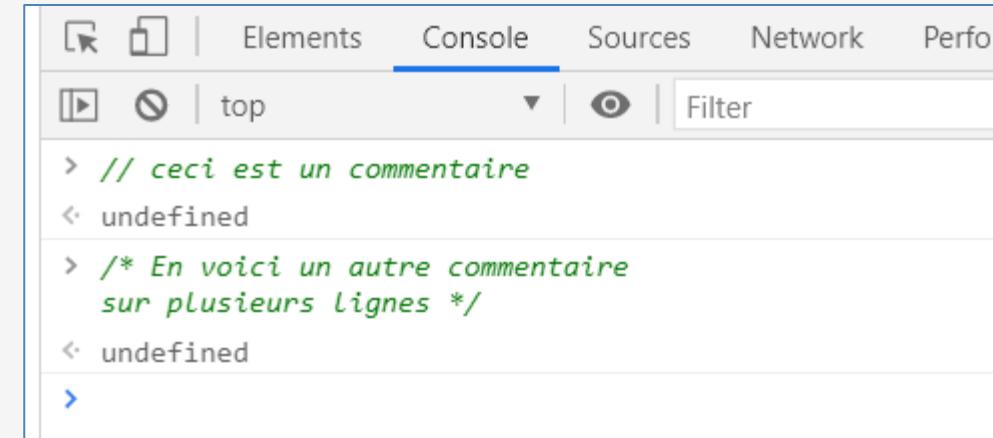
1. Déclaration et portée des variables.
2. Types de données (Number, Boolean, Date, Math, String, Array).
3. Conversion de type. Détection de type avec typeof.
4. Opérateurs logiques et arithmétiques. Gestion des tableaux. Boucles (for, while...).
5. Création de fonctions et paramétrage variable.
6. Faire un codage sécurisé avec la gestion d'erreur et les exceptions.
7. Rappels sur les concepts objets. Développement Objet en JavaScript : création de classes (méthodes, propriétés).
8. Constructeur. Surcharge. Mots réservés prototype, this. Crédit d'instance. Usage d'Object sur les classes dynamiques.
9. Utilisation du format JSON pour la création de classes.
10. Les objets prédéfinis du langage (Array, Date, String, Regexp...) et leur utilisation. Extension des objets prédéfinis.

Exercices d'application

1.1- Déclaration et portée des variables

Voir codes sur https://github.com/bassemSeddik/INETUM_JavaScript

- Les instructions se terminent pas un ";"
- Les commentaire sur une ligne commence par "://"
- Les commentaires sur plusieurs lignes sont entre "/*" et "*/"
- Les blocs sont délimités par "{" et "}"



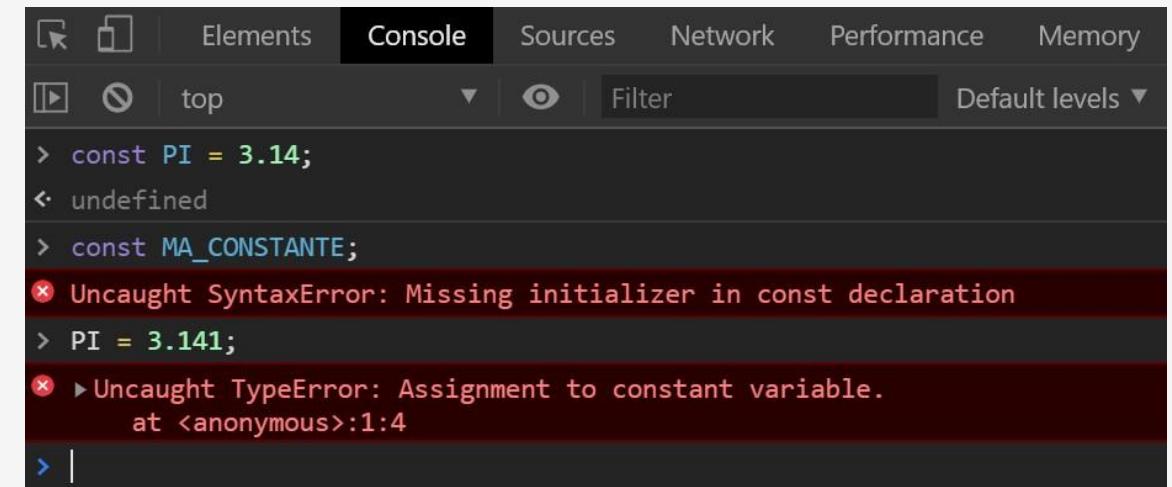
```

> // ceci est un commentaire
< undefined
> /* En voici un autre commentaire
   sur plusieurs lignes */
< undefined
>

```

Constante:

- Mot clé : **const**
- Nécessite une initialisation au moment de la création
- Ne peut pas changer de valeur après initialisation



```

> const PI = 3.14;
< undefined
> const MA_CONSTANTE;
✖ Uncaught SyntaxError: Missing initializer in const declaration
> PI = 3.141;
✖ ▶ Uncaught TypeError: Assignment to constant variable.
  at <anonymous>:1:4
>

```

1.1- Déclaration et portée des variables (suite)

Variables locales

- Mot clé: **let**

```
> console.log(nom)
✖ > Uncaught ReferenceError: nom is not defined
      at <anonymous>:1:13
> let nom = 'Bassem'
< undefined
> console.log(nom)
Bassem
> {
    let ville = 'Sousse';
}
< undefined
> console.log(ville)
✖ > Uncaught ReferenceError: ville is not defined
      at <anonymous>:1:13
```

Variables globales

- Mot clé: **var**

```
> nom
✖ > Uncaught ReferenceError: nom is not defined
      at <anonymous>:1:1
> var nom = 'Seddik'
< undefined
> nom
< "Seddik"
> {
    var prenom = 'Bassem'
}
< undefined
> prenom
< "Bassem"
```

1.2- Types de données (Number, Boolean, Date, Math, String, Array). Conversion de type.

En JavaScript il y a cinq différents types de données susceptibles de contenir des valeurs :

- string
- number
- boolean
- object
- function

Il y a six types d'objets :

- Object
- Date
- Array
- String
- Number
- Boolean

Il y a deux types qui ne peuvent pas contenir des objets :

- null
- undefined

Voir codes sur [/INETUM JavaScript \(II- PriseEnMains JS\Les variables JS\)](#)

```
// Ceci est un commentaire sur une ligne
/*
C'est un commentaire sur
plusieurs lignes
*/
var x = 2;
var y = 2.5;
var x1 = "hello", y1 = 'hi', z1 = 3;
var yes = true, X = false;
var Y = null;
alert (" la varible x est: " + x + ", elle est de type " + typeof(x));
alert (" la varible y est: " + y + ", elle est de type " + typeof(y));
alert (" la varible x1 est: " + x1 + ", elle est de type " + typeof(x1));
alert (" la varible yes est: " + yes + ", elle est de type " + typeof(yes));
alert (" la varible Y est: " + Y + ", elle est de type " + typeof(Y));
alert('Salut les membres "ORSYS"');
```

Chaines de caractères

- Les chaînes de caractères sont des valeurs composées de texte et peuvent contenir des lettres, des nombres, des symboles, de la ponctuation, et même des emoji.
- Elles sont contenues dans une paire de guillemets 'simples' ou de "doubles" guillemets.
- Example pour inclure des guillemets dans le texte:

```
"It's six o'clock.";  
'Remember to say "please" and "thank you."';
```

```
var blague = "Au confinement, c'est  
exactement comme à l'age de 16!";  
blague += "Eh bien, vous avez cheveux long, ...  
tout est moins cher, ... et puis vous êtes interdis  
de sortir!";  
console.log(blague);
```

EXAMPLE

```
'This is a string. 🙌';  
"This is the 2nd string. 🧑";
```

EXAMPLE

```
'It\'s six o\'clock.';  
"Remember to say \"please\" and \"thank you.\"";
```

Exercice:

Définissez deux variables appelées **thingOne** et **thingTwo** et affectez-leur des valeurs.

Imprimez les valeurs des deux variables dans une seule instruction **console.log** à l'aide de la concaténation. Par exemple,

rouge Bleu

où "rouge" est la valeur de thingOne et "bleu" est la valeur de thingTwo. N'oubliez pas d'utiliser un point-virgule à la fin de chaque instruction!

```
var thingOne = "red";  
var thingTwo = "blue";  
console.log(thingOne + " " + thingTwo);
```

1.3- Détection de type avec typeof

L'opérateur `typeof` renvoie une chaîne qui indique le type de son opérande.

Type	Résultat
indéfini	"undefined"
nul	"object"
booléen	"boolean"
nombre	"number"
grand entier	"bigint"
chaîne de caractère	"string"
symbole (nouveauté d'ECMAScript 6 / 2015)	"symbol"
objet de l'environnement (fourni par l'environnement dans lequel est utilisé JS)	Résultat différent selon l'implémentation
Objet Function (au sens ECMA-262, un objet qui implémente [[Call]])	"function"
Tout autre objet	"object"

```
console.log(typeof 42);  
// expected output: "number"  
  
console.log(typeof 'blubber');  
// expected output: "string"  
  
console.log(typeof true);  
// expected output: "boolean"  
  
console.log(typeof undeclaredVariable);  
// expected output: "undefined";
```

Conditions d'égalité entre types

- `==` égalité en valeur
- `====` égalité en valeur et en type

```
console.log ( 4 == "4" );
console.log ( 4 === "4" );
console.log ( 4 != "4" );
console.log ( 4 !== "4" );
var y=10;
console.log ( "la variable y est " + y + " elle est de type " + typeof(y));
var y="10";
console.log ( "la variable y est " + y + " elle est de type " + typeof(y));
```

The screenshot shows a browser's developer tools console window. At the top, there are icons for play/pause, stop, and refresh, followed by 'top' and a dropdown menu. Below that is a 'Filter' input field. The console output area contains the following text:

```
true
false
false
true
la variable y est 10 elle est de type number
la variable y est 10 elle est de type string
>
```

Demo: Combinaison entre HTML et JavaScript

1. HTML

```
<body>
<h1>Les chaines de caracteres</h1>
<p>Un paragraphe</p>
<p id='p1'></p>
<p id='p2'></p>
<p id='p3'></p>
<p id='p4'></p>
<p id='p5'></p>
</body>
```

2. JavaScript

```
let prenom = "Je m'appelle Bassem";
let age = 29;
let age2 = '29';

document.getElementById('p1').innerHTML = 'Type de prenom : ' + typeof prenom;
document.getElementById('p2').innerHTML = 'Type de age : ' + typeof age;
document.getElementById('p3').innerHTML = 'Type de age2 : ' + typeof age2;

// Les types spéciaux
let nullVariable = null;
let undefinedVariable;

document.getElementById('p4').innerHTML = 'Type de nullVariable : ' + typeof
nullVariable;
document.getElementById('p5').innerHTML = 'Type de undefinedVariable : ' + typeof
undefinedVariable;
```

1.4- Opérateurs logiques et arithmétiques.

Gestion des tableaux. Boucles (for, while...).

Un opérateur est un symbole qui va être utilisé pour effectuer certaines actions notamment sur les variables et leurs valeurs.

Opérateur	Nom de l'opération associée
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

```
x = 1 - 2 - 3; //Calcule (1 - 2) - 3 = 1 - 3 = - 4  
y = 1 - (2 - 3); //Calcule 1 - (2 - 3) = 1 - (- 1) = 1 + 1 = 2  
z = 2 ** 3 ** 2; //Calcule 3 ** 2 = 3 * 3 = 9 puis 2 ** 9 = 512
```

```
var x = 5, y=10, z=-2;  
x=x+1;  
x+=1;  
x=x-2;  
x-=z;  
y=y*2;  
y*=2;  
var multi = x*y;  
var divi = y/3;  
var mod = 13%3;  
z = x+y/(4+z)%3;  
  
alert ("z est " + z);
```

Exercice: Sortons faire un diner!

1. Créez une variable appelée **facture** et affectez-lui le résultat de **10,25 + 3,99 + 7,15** (ne faites pas le calcul vous-même, laissez JavaScript le faire!).
2. Ensuite, créez une variable appelée pourboire et affectez-lui le résultat de la multiplication de la facture par un taux de pourboire de **15%**.
3. Enfin, ajoutez la facture et le pourboire ensemble et stockez-les dans une variable appelée total.
4. Affichez le total sur la console JavaScript.

```
var bill = 10.25 + 3.99 + 7.15;  
var tip = 0.15 * bill;  
var total = bill + tip;  
console.log("$"+total.toFixed(2));  
// You can simply print the  
total, without the currency  
symbol //  
console.log(total);
```

1.4- Opérateurs logiques et arithmétiques.

Gestion des tableaux. Boucles (for, while...).

- Objectif : Stocker plusieurs valeurs dans une seule variable

```
var chiffres = [5, 8, 28];
var prenoms = ['Pierre', 'Victor', 0.1, 'Claire'];
var lettres = new Array('c', 'f', 'g');

alert(prenoms); alert(typeof(prenoms));
alert(chiffres[2]); alert(typeof(chiffres));
alert(lettres[1]); alert(typeof(lettres));
```

- Parcourir son tableau

```
var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'], p="";
for(var i = 0; i < prenoms.length; i++){
    p += 'Prénom n°' + (i+1) + ':' + prenoms[i] + '\n';
}
alert(p);
```

- affectation **prenoms[0] = 'Paul';**
- Rajout en fin de tableau
prenoms[prenoms.length] = 'Chloé';
- Tableaux associatifs (par clé au lieu d'index)

```
//On crée un objet littéral
var prenoms = {
    prenom1 : 'Pierre',
    prenom2 : 'Victor',
    prenom3 : 'Julia',
};

var p = '';
for (var clefs in prenoms){
    p += clefs + ':' + prenoms[clefs] + '\n';
}
```

Boucles (suite)

```
var x = 20;
do {
    console.log ("x contient la valeur : " + x);
    x--;
}
while (x > 10)
```

```
x contient la valeur : 20
x contient la valeur : 19
x contient la valeur : 18
x contient la valeur : 17
x contient la valeur : 16
x contient la valeur : 15
x contient la valeur : 14
x contient la valeur : 13
x contient la valeur : 12
x contient la valeur : 11
```

```
var x = 0;
while (x < 10) {
    console.log('x contient la valeur : ' + x);
    x++; //Equivaut à x = x + 1 ou x += 1
}
```

```
x contient la valeur : 0
x contient la valeur : 1
x contient la valeur : 2
x contient la valeur : 3
x contient la valeur : 4
x contient la valeur : 5
x contient la valeur : 6
x contient la valeur : 7
x contient la valeur : 8
x contient la valeur : 9
```

Clauses Conditionnelles

- if/else :

```
if(somme <1000) {  
    remise = 0;  
} else if (somme <2000) {  
    remise = 5;  
}else{  
    remise 10;  
}
```

- switch :

```
switch (new Date().getDay()) {  
    case 0:  
    case 6:  
        console.log("Fin de semaine");  
        break;  
    default:  
        console.log("Jour de semaine")  
}
```

- Condition ternaire

```
let test = isNaN(val) ? "Ce n'est pas une valeur numérique" : "C'est une valeur numérique";
```

1.5- Crédation de fonctions et paramétrage variable

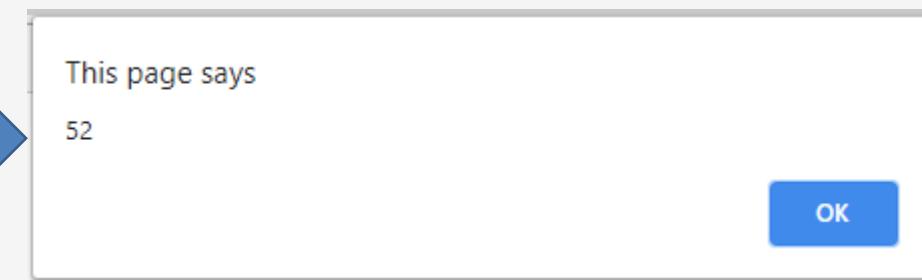
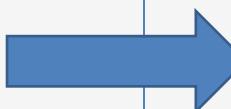
Cela permet, comme pour toute fonction Classique, de ne pas avoir à refaire le même bout de code encore et encore à chaque besoin d'un même traitement!

- Il suffit de définir sa fonction et d'y faire appel quand voulu,
- Bien sûr, il faut le faire avec les bons paramètres!

```
function nomDeFonction( liste de paramètres séparés par des virgules)
{
    instructions;
}
```

```
function multiplication(x, y){
    alert(x*y);
}

multiplication(26, 2);
multiplication(-4, 60);
multiplication(3.14, 3.14);
```



Fonction: Valeur de retour

- Si vous vous rappelez, nous avons déjà utilisé la fonction `console.log()`, nous avons trouvé un message “**undefined**” qui apparaissait toujours après notre résultat!
 - C'est la valeur de retour de notre fonction
 - Toute fonction est sensée retourner une valeur!



```
1  function reverseString(reverseMe) {  
2      var reversed = "";  
3      for (var i = reverseMe.length - 1; i >= 0; i--) {  
4          reversed += reverseMe[i];  
5      }  
6      return reversed;  
7  }  
8  
9  console.log(reverseString("Julia"));  
10
```

“ailuJ”



Fonction: options possibles

Voir codes sur [INETUM JavaScript \(II- PriseEnMains JS/05 Les fonctions JS\)](#)

- Fonction avec des **paramètres variables**:

- **vars** est traité comme un tableau et doit être le dernier paramètre

```
function nomDeFonction([param,...] vars){  
    instructions  
}
```

- Fonction **anonyme**:

```
let f = function () {  
    console.log("Je suis une fonction");  
};  
  
f();
```

- Appel immédiat d'une fonction anonyme (**closures**):

```
(function () {  
    console.log("Je suis une fonction");  
})();
```

```
function somme(... valeurs){  
    let somme = 0;  
    for (x in valeurs){  
        somme += valeurs[x];  
    }  
    return somme;  
}  
alert ("somme(2,3,4)=" + somme(2,3,4) + " et  
somme(20,30)=" + somme(20,30));
```

Fonction: options possibles (suite)

Voir codes sur [INETUM JavaScript](#) (II- PriseEnMains JS/05 Les fonctions JS)

- Fonctions fléchées:

```
// On va utiliser une fonction! Oui le signe () le confirme
// Et, pas la peine de lui donner un nom!
// Son code? le voici, juste après la flèche
let getNombre = () => { return Math.random(); }
alert(getNombre());
```

```
let GetRandomMessage = (prefixe, suffixe) => {
    let number = Math.random();
    return prefixe + number + suffixe;
}
alert(GetRandomMessage("<< ", " >>"));
```

- Appel aux paramètres par défaut:

```
let affichePrix = function(valeur, devise = 'Euro') {
    console.log(`Prix : ${valeur} ${devise}`)
};

affichePrix(100, 'dollars');
affichePrix(200);
```

Exercice d'application (tableaux et fonctions)

- On cherche à s'authentifier et accéder à notre page Web avec le bon mot de passe.
- Pour cela, nous allons donner 3 chances pour saisir le bon **password** parmi une liste de passwords pré-déclarées dans un tableau **PasswordS**.
- Pour donner 3 chances nous avons besoin d'une boucle exploitant la fonction

```
var variable = prompt("message");
```

- Pour comparer notre variable obtenue avec toutes les cases du tableau, nous allons déclarer une fonction dédié:
- ```
function VerifierExistancePassword(pwd, listePWDs)
```
- Proposez à la fin un équivalent en notation « fléchée » à votre fonction

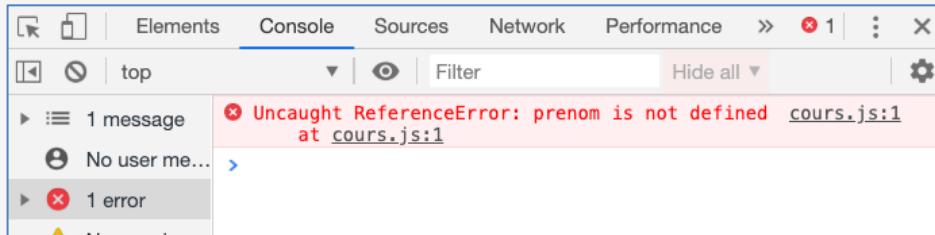
```
var PasswordS = ['123', 'abc', 'ddd'];
var pwd;
var i=1;
// Déclaration de la fonction de vérification
function VerifierExistancePassword (pwd, listePWDs) {
 for (// TODO){
 if (// TODO){
 return true;
 }
 }
 return false;
}

// Nous allons donner 3 chances pour donner le bon mot de passe
do{
 pwd = prompt("Donnez le mot de passe");
 // Appel à une fonction dédiée
 if (// TODO){
 // TODO
 break;
 }
 i++;
}
while (// TODO)
```

# 1.6- Faire un codage sécurisé avec la gestion d'erreur et les exceptions.

On peut appliquer une gestion des erreurs sous JavaScript:

- Gérer une erreur avec les blocs **try...catch**



## Exemple:

- Dans le cas où un utilisateur tente de diviser par zéro, une erreur va donc être provoquée.
- On va pouvoir prendre en charge ce cas précis en amont en lançant (**throw**) ce qu'on appelle une exception si besoin
- Une Exception est un objet instanciable par **new Error(...)**
- Le bloc **finally** est un bloc optionnel qui doit être placé juste après un try...catch

```
function divFunction(){
 let x = prompt('Entrez un premier nombre (numérateur)');
 let y = prompt('Entrez un deuxième nombre (dénominateur)');

 if(isNaN(x) || isNaN(y) || x == "" || y == ""){
 throw new Error('Merci de rentrer deux nombres');
 }else if(y == 0){
 throw new Error('Division par 0 impossible')
 }else{
 alert("Le résultat de la division est: " + x / y);
 }

 try{
 divFunction();
 }catch(err){
 alert(err.message);
 }finally{
 alert('Ce message s\'affichera quoiqu\'il arrive');
 }
}
```

# Exercice

- On souhaite récupérer une moyenne comprise en 0 et 20 de l'utilisateur,
  - puis on calcule la mention (En dessous de la moyenne, passable, Assez bien, Bien et Très bien),
  - et enfin afficher la mention à l'utilisateur

Cette page indique

Donnez une moyenne entre 0 et 20

OK Annuler

Cette page indique

14 : Bien

OK

- **Problème** : L'utilisateur peut fournir n'importe quelle valeur ou même rien
  - Utilisez la gestion des Exceptions

Le mode strict de JavaScript est, comme son nom l'indique, un mode qui va contenir une sémantique légèrement différente et plus stricte par rapport au JavaScript « classique ».

### use strict

- Exemple 1: Avec le mode strict, cela ne sera pas permis : si vous omettez **let** (ou **var**) dans la déclaration d'une variable, une erreur va être lancée et le script ne s'exécutera pas!
- Exemple 2: En mode strict, la méthode eval() n'est pas autorisée à créer des variables dépassant sa propre portée

```
<body>
 <h1>Le mode strict</h1>
 <p id='varX'></p>
 <p id='varY'></p>
 <script>
 var afficheX = document.getElementById('varX');
 var afficheY = document.getElementById('varY');

 function affX(){
 'use strict';
 var x = 5;
 alert(x);
 }

 function affY(){
 'use strict';
 y = 10;
 alert(y);
 }

 affX();
 affY();
 </script>
</body>
```

# 1.7-Développement Objet en JavaScript : création de classes (méthodes, propriétés).

Voir codes sur [formationDHL](#) (II- PriseEnMains JS/06 Les Objets JS

Rappel sur les Array (s)

Le Array est l'une des structures de données les plus utiles en JavaScript

```
const myArray = [];
```

Chaque élément d'un tableau est référencé par une clé numérique appelée index, qui commence à zéro et incrémenté de un pour chaque élément supplémentaire du tableau.

```
const fruits = ['apple', 'banana', 'orange', 'grape', 'lychee'];

console.log(fruits);
// ['apple', 'banana', 'orange', 'grape', 'lychee']
```

Si nous voulons récupérer le premier élément (le plus à gauche) dans fruits, nous accédons à cet élément par son index:

```
fruits[0];
// 'apple'
```

# Les objets en JavaScript

L'objet est l'une des structures de données les plus importantes de JavaScript. Après tout, vous suivez actuellement un cours complet sur la programmation orientée objet!

Fondamentalement, un objet est une collection de paires clé/valeur associées. Nous créons un objet avec des accolades { et }. Voici une variable appelée myObject, qui est affectée à un objet vide:

```
const myObject = {};
```

Alors que les éléments des tableaux sont référencés par un index numérique, les clés d'un objet doivent être nommées explicitement, comme la couleur ou l'année.

```
const car = {
 color: 'red',
 year: 1992,
 isPreOwned: true
};
```

# QUIZ

---

Lesquels des éléments suivants sont des caractéristiques d'un objet? Sélectionnez tout ce qui s'y rapporte:

- A. Ordonné
- B. Non-ordonné
- C. Couple clé/valeur
- D. Indexé
- E. Accollades { ... }
- F. Crochets [ .....

Décomposons cela et voyons ce qui se passe:

- La variable affectée à l'objet est nommée car.
- Les accolades sont utilisées pour définir l'objet car.
- Les clés individuelles (exp. color) sont associées à une seule valeur («'red'» dans ce cas). Ces paires clé/valeur sont connectées par deux points (:).
- Chaque paire clé/valeur distincte, connue sous le nom de propriété de cet objet, est séparée des autres propriétés par une virgule (,). L'objet car contient donc trois propriétés.

L'ordre dans lequel apparaissent les variables n'est pas important, on peut le changer sans que cela impact les index. Pas comme dans les index des Arrays.

# Syntaxe d'écriture des propriétés des objets

- Il est à noter que les clés (les noms des propriétés de l'objet) sont des chaînes, mais les guillemets entourant ces chaînes sont facultatifs
- tant que la chaîne est également un identifiant Javascript valide c'est-à-dire que vous pouvez l'utiliser comme nom de variable ou nom de la fonction
- Par conséquent, les trois objets suivants sont équivalents:

```
const course = { courseId: 711 }; // ← no quotes around the courseId key
const course = { 'courseId': 711 }; // ← single quotes around the courseId key
const course = { "courseId": 711 }; // ← double quotes around the courseId key
```

# Exercice

- Créez un objet appelé « menu » qui représente l'élément de menu suivant :
  - Salted Caramel Ice Cream
  - 2.95
  - butter, ice cream, salt, sugar
- L'objet doit contenir les propriétés suivantes:
  - name
  - price
  - ingredients
- Associez des valeurs appropriées à chaque attribut.

## Solution à l'exercice :

```
menu = {
 name: 'Salted Caramel Ice Cream',
 price: 2.95,
 ingredients: ['butter', 'ice cream', 'salt', 'sugar']
};
```

# Accès aux propriétés des objets

Comment pouvons-nous en extraire des informations depuis les objets? En d'autres termes: comment accédons-nous à leurs valeurs?

- Il existe deux façons: la notation par points et la notation par crochets. Considérez cet objet **bicycle**:

- La notation par point .

```
bicycle.color;
// 'blue'
```

```
bicycle['color'];
// 'blue'
```

- Aussi, nous pouvons accéder à la même propriété en utilisant la notation entre crochets

- Et pour les objets imbriqués?

**Exercice** ([voir lien](#))

```
bicycle.wheels.width;
// 8
bicycle['wheels']['width'];
// 8
```

```
const bicycle = {
 color: 'blue',
 type: 'mountain bike',
 wheels: {
 diameter: 18,
 width: 8
 }
};
```

# Objets instances et JSON

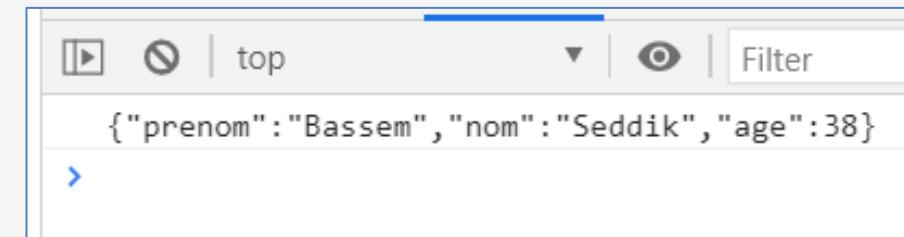
- Les Objet literal sont une creation directe d'une instance à une classe
- Le format est conforme à la representation JSON : JavaScript Object Notation

```
let maVoiture3 = {
 id: 300,
 style: 'SUV'
};

console.log(JSON.stringify(maVoiture3));
```

```
var moi = {
 prenom: "Bassem",
 nom: "Seddik",
 age: 38,
 //this sert de référence à l'objet utilisé
 id: function(){
 return this.prenom + ' ' + this.nom;
 }
};

//On affiche le résultat retourné par notre méthode
alert(typeof(moi));
alert(moi.id());
console.log(JSON.stringify(moi));
```



# Fonctions JavaScript first-class

En JavaScript, les fonctions sont des fonctions dites First-class.

```
function Voiture (id){
 this.idVoiture = id;
 this.demarrer = function(){
 console.log('Démarrage de ' + this.idVoiture);
 }
}
let maVoiture = new Voiture(133);
maVoiture.demarrer();
```

On peut accéder à la propriété **prototype** de tout objet et appliquer des ajouts

```
Voiture.prototype.arret = function(){
 console.log ("ARRET de " + this.idVoiture + "!");
}
maVoiture.arret();
```

# Classes comme sous les autres langages (OOP)!

Voir codes sur [formationDHL](#) (II- PriseEnMains JS/06 Les Objets JS)

1. Définition d'une classe
2. Définition de constructeurs
3. Définition de propriétés
4. Définition de méthodes
5. Héritage

```

 | top
 {"idVoiture":133}
 ID voiture: 209
 {"idVoiture":588}
 ID voiture: 2009
 300
> |

```

```

class Voiture{
 constructor(id){
 this.idVoiture = id;
 }
 getInfo(){
 return `ID voiture: ${this.idVoiture}`;
 }
}

```

```

class VoitureDeSport extends Voiture{
 constructor(id, max){
 super(id);
 this.maxVitesse = max;
 }
 getVitesseMax(){
 return this.maxVitesse;
 }
}

```

```

let maVoiture1 = new Voiture();
let maVoiture2 = new Voiture(133);
let maVoiture3 = new Voiture(209);
console.log(JSON.stringify(maVoiture2));
console.log(maVoiture3.getInfo());

```

```

let maSuperVoiture1 = new VoitureDeSport();
let maSuperVoiture2 = new VoitureDeSport(588);
let maSuperVoiture3 = new VoitureDeSport(2009, 300);
console.log(JSON.stringify(maSuperVoiture2));
console.log(maSuperVoiture3.getInfo());
console.log(maSuperVoiture3.getVitesseMax());

```

# 1.8- Les objets prédéfinis du langage (Array, Date, String, Regexp...) et leur utilisation. Extension des objets prédéfinis.

L'objet ***Navigator*** : votre navigateur. En voici des propriétés, méthodes et objets :

```
1.Navigator {...}
 1.appCodeName: (...)
```

```
 2.appName: (...)
```

```
 3.appVersion: (...)
```

```
 4.bluetooth: (...)
```

```
 5.clearAppBadge: f clearAppBadge()
```

```
 6.clipboard: (...)
```

```
 7.connection: (...)
```

```
 8.cookieEnabled: (...)
```

```
 9.credentials: (...)
```

```
10.deviceMemory: (...)
```

```
11.doNotTrack: (...)
```

```
12.geolocation: (...)
```

```
13.getBattery: f getBattery()
```

```
14.getGamepads: f getGamepads()
```

```
15.getInstalledRelatedApps: f getInstalledRelatedApps()
```

```
16.getUserMedia: f getUserMedia()
```

```
17.hardwareConcurrency: (...)
```

```
18.javaEnabled: f javaEnabled()
```

```
19.keyboard: (...)
```

```
20.language: (...)
```

```
21.languages: (...)
```

```
22.locks: (...)
```

```
23.maxTouchPoints: (...)
```

```
24.mediaCapabilities: (...)
```

```
25.mediaDevices: (...)
```

```
26.mediaSession: (...)
```

```
27.mimeTypes: (...)
```

```
28.onLine: (...)
```

```
29.__proto__: Object
```

```
...
```

# 1.8- Les objets prédéfinis du langage (Array, Date, String, Regexp...) et leur utilisation. Extension des objets prédéfinis.

L'objet **Date** : La date et l'heure sont regroupées en JS dans la classe Date :

*Date.prototype*

1.**constructor**: f Date()  
2.**getDate**: f getDate()  
3.**getDay**: f getDay()  
4.getFullYear: f getFullYear()  
5.**getHours**: f getHours()  
6.getMilliseconds: f getMilliseconds()  
7.getMinutes: f getMinutes()  
8.getMonth: f getMonth()  
9.**getSeconds**: f getSeconds()  
10.**getTime**: f getTime()  
11.getTimezoneOffset: f getTimezoneOffset()  
12.**getUTCDate**: f getUTCDate()  
13.getUTCDay: f getUTCDay()  
14.getUTCFullYear: f getUTCFullYear()  
15.getUTCHours: f getUTCHours()  
16.getUTCMilliseconds: f getUTCMilliseconds()  
17.getUTCMinutes: f getUTCMinutes()

18.getUTCMonth: f getUTCMonth()  
19.getUTCSeconds: f getUTCSeconds()  
20.**getYear**: f getYear()  
21.setDate: f setDate()  
22.setFullYear: f setFullYear()  
23.setHours: f setHours()  
24.setMilliseconds: f setMilliseconds()  
25.**setMinutes**: f setMinutes()  
26.setMonth: f setMonth()  
27.**toString**: f toString()  
28.toTimeString: f toTimeString()  
29.toUTCString: f toUTCString()  
30.valueOf: f valueOf()  
31.Symbol(Symbol.toPrimitive): f [Symbol.toPrimitive]()  
32.\_\_proto\_\_: Object  
33....

# 1.8- Les objets prédéfinis du langage (Array, Date, String, Regexp...) et leur utilisation. Extension des objets prédéfinis.

L'objet **String** : Pour la manipulation de chaînes de caractères:

`String.prototype`

1.anchor: *f* anchor()  
2.blink: *f* blink()  
**3.bold:** *f* bold()  
**4.charAt:** *f* charAt()  
5.charCodeAt: *f* charCodeAt()  
6.codePointAt: *f* codePointAt()  
**7.concat:** *f* concat()  
**8.constructor:** *f* String()  
9.endsWith: *f* endsWith()  
10.fixed: *f* fixed()  
11.fontcolor: *f* fontcolor()  
12.fontsize: *f* fontsize()  
13.includes: *f* includes()  
14.indexOf: *f* indexOf()  
15.italics: *f* italics()  
16.lastIndexOf: *f* lastIndexOf()  
**17.length:** *0*

18.repeat: *f* repeat()  
**19.replace:** *f* replace() startsWith: *f* startsWith()  
20.strike: *f* strike()  
21.sub: *f* sub()  
22.toLowerCase: *f* toLowerCase()  
23.toString: *f* toString()  
24.toUpperCase: *f* toUpperCase()  
**25.trim:** *f* trim()  
26.trimEnd: *f* trimEnd()  
27.trimLeft: *f* trimStart()  
28.trimRight: *f* trimEnd()  
29.trimStart: *f* trimStart()  
30.valueOf: *f* valueOf()  
31.Symbol(Symbol.iterator): *f* [Symbol.iterator]()  
32.\_\_proto\_\_: Object  
...

# 1.8- Les objets prédéfinis du langage (Array, Date, String, Regexp...) et leur utilisation. Extension des objets prédéfinis.

L'objet **RegExp** : Application de contrôles sur les contenus des expressions:

*RegExp.prototype*  
1.compile: *f compile()*  
**2.constructor:** *f RegExp()*  
3.dotAll: (...)  
**4.exec:** *f exec()*  
5.flags: (...)  
6.global: (...)  
7.ignoreCase: (...)  
8.multiline: (...)  
9.source: (...)  
10.sticky: (...)  
**11.test:** *f test()*  
12.toString: *f toString()*  
13.unicode: (...)  
14.Symbol(Symbol.match): *f [Symbol.match]()*  
15.Symbol(Symbol.matchAll): *f [Symbol.matchAll]()*

16.Symbol(Symbol.replace): *f [Symbol.replace]()*  
17.Symbol(Symbol.search): *f [Symbol.search]()*  
18.Symbol(Symbol.split): *f [Symbol.split]()*  
19.get dotAll: *f dotAll()*  
20.get flags: *f flags()*  
21.get global: *f global()*  
**22.get ignoreCase:** *f ignoreCase()*  
23.get multiline: *f multiline()*  
24.get source: *f source()*  
25.get sticky: *f sticky()*  
26.get unicode: *f unicode()*  
27.\_\_proto\_\_: Object

## Méthodes de l'objet Array

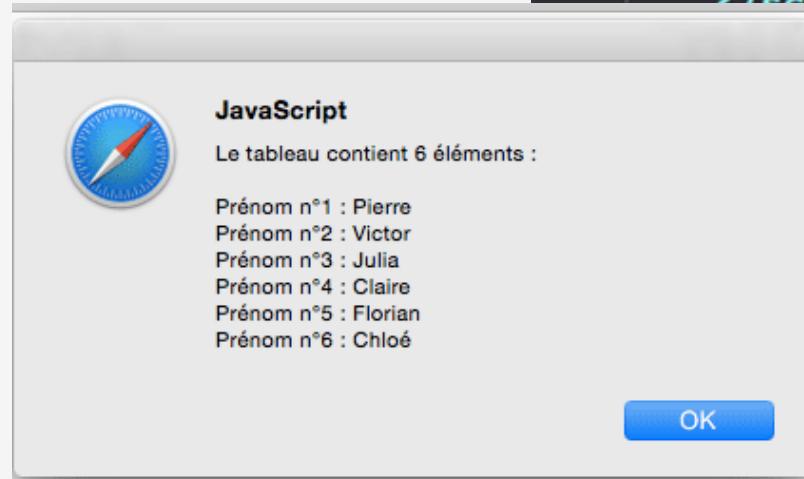
- Pour ajouter des éléments en fin de tableau, on va utiliser la méthode **push()**: va retourner la nouvelle taille du tableau
- Pour supprimer des éléments en fin de tableau, on va utiliser **pop()**: va retourner dans une chaîne de caractères la valeur relative à l'élément supprimé.
- Les méthodes **shift()** et **unshift()** ;
- La méthode **splice()** ;
- La méthode **sort()** ;
- La méthode **reverse()** ;
- La méthode **join()** ;
- La méthode **slice()** ;
- La méthode **concat()**.

La variable **taille** nous sert à récupérer l'information renvoyée par la méthode `push()`: taille du nouveau tableau.

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'], p = '';
 //On ajoute les valeurs Florian et Chloé en fin de tableau
 var taille = prenoms.push('Florian', 'Chloé');

 //On affiche les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }

 //On affiche la taille du tableau et les prénoms
 alert('Le tableau contient ' +taille+ ' éléments : \n\n' +p);
 </script>
```



# Exemple d'application

- Création d'un tableau de transactions
- Définition d'une function init() qui y rajoute
  - Nos entrées
  - Nos dépense
- Appel de la function init()

```
var transactions = [];

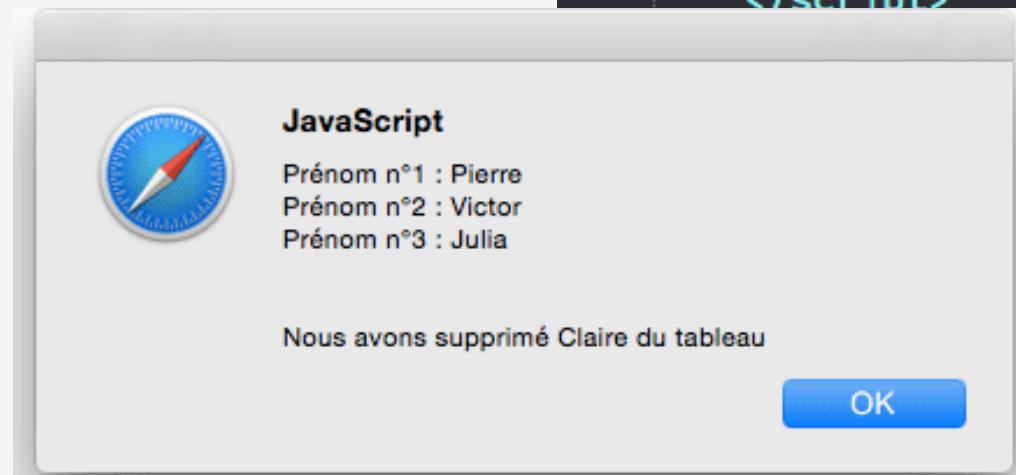
// Init app
function init() {
 transactions.push("ENTREE: virement depuis un client ");
 transactions.push("SORTIE: payer facture ");
 transactions.push("SORTIE: acheter un pc");
 transactions.push("SORTIE: payer facture de gaz ");
 console.log(transactions);

 // ici des exemple de gestion du tableau
}

// call init
init();
```

La variable **suppr** va récupérer l'information renvoyée par **p**, valeur qui correspond à l'élément supprimé.

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'], p = '';
 //On supprime la dernière valeur du tableau
 var suppr = prenoms.pop();
 //On affiche les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }
 //On affiche les prénoms du tableau et le prénom supprimé
 alert(p + '\n\nNous avons supprimé ' +suppr+ ' du tableau');
 </script>
```

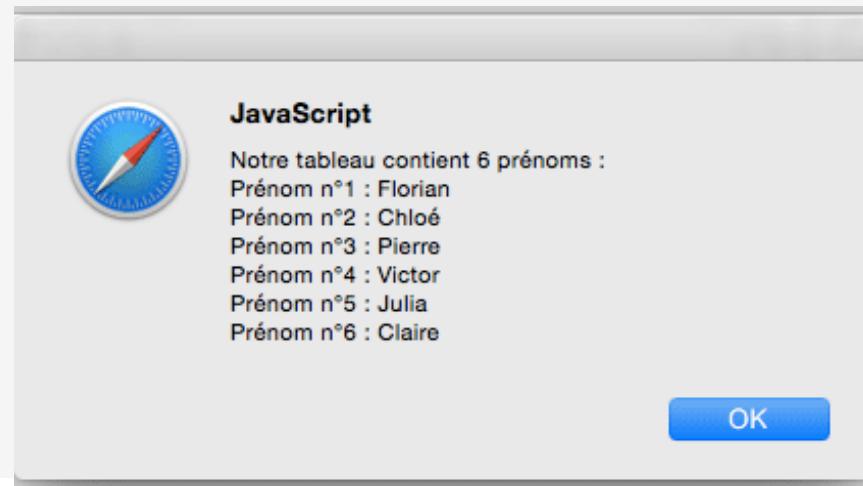


# Ajouter et supprimer des éléments au début

Ajouter ou supprimer des éléments en début de tableau, on va respectivement utiliser les méthodes **unshift()** et **shift()**

**unshift()** va, comme **push()**, retourner la nouvelle taille du tableau

**shift()**, comme **pop()**, va retourner la valeur supprimée sous forme de chaîne de caractères.



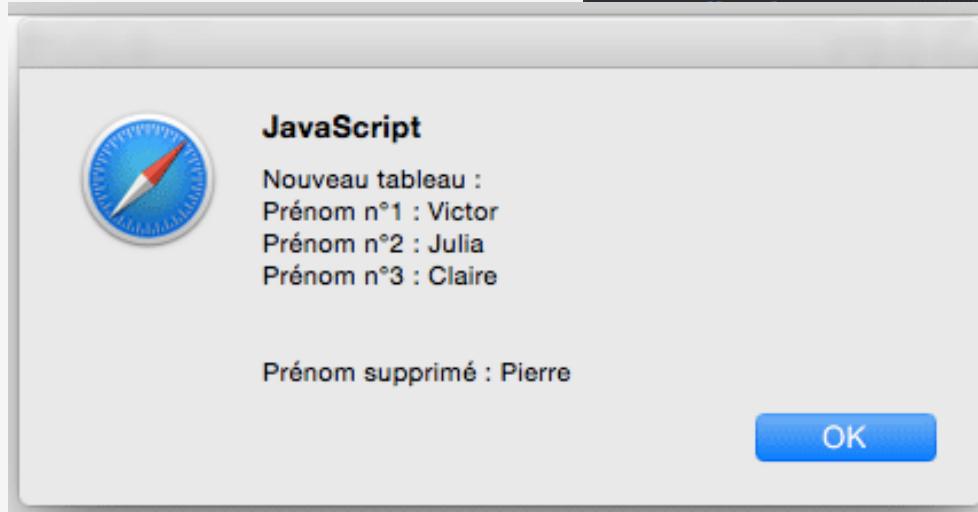
```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'], p = '';

 //On ajoute deux éléments en début de tableau
 var taille = prenoms.unshift('Florian', 'Chloé');

 //On affiche les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }

 //On affiche les prénoms du tableau et sa taille
 alert('Notre tableau contient ' +taille+ ' prénoms :\n' +p);
 </script>
 </body>
</html>
```

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'], p = '';
 //On supprime le premier prénom du tableau
 var suppr = prenoms.shift();
 //On récupère les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }
 //On affiche les prénoms du tableau et le prénom supprimé
 alert('Nouveau tableau :\n' +p+ '\n\nPrénom supprimé : ' +suppr);
 </script>
```



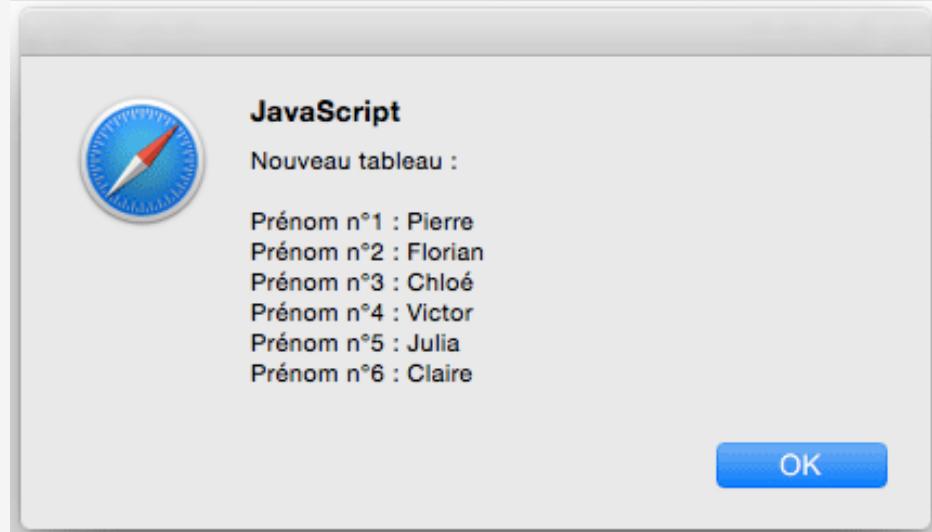
# Ajouter ou supprimer des éléments choisis

- La méthode **splice()** nous permet d'ajouter ou de supprimer un élément n'importe où dans un tableau
- prend au minimum deux arguments :
  - la position à partir de laquelle les éléments seront ajoutés
  - combien d'éléments doivent être enlevés.
- les nouveaux éléments vont se placer juste avant l'élément correspondant à l'indice donné
- Après ces deux arguments obligatoires, on peut rajouter autant d'autres arguments qu'on veut insérer de valeurs

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'], p = '';
 /*On ajoute 2 éléments après 'Pierre' et avant 'Victor'.
 On ne supprime aucun élément/
 prenoms.splice(1, 0, 'Florian', 'Chloé');

 //On récupère les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }

 //On affiche les prénoms du tableau
 alert('Nouveau tableau :\n\n' +p);
 </script>
```

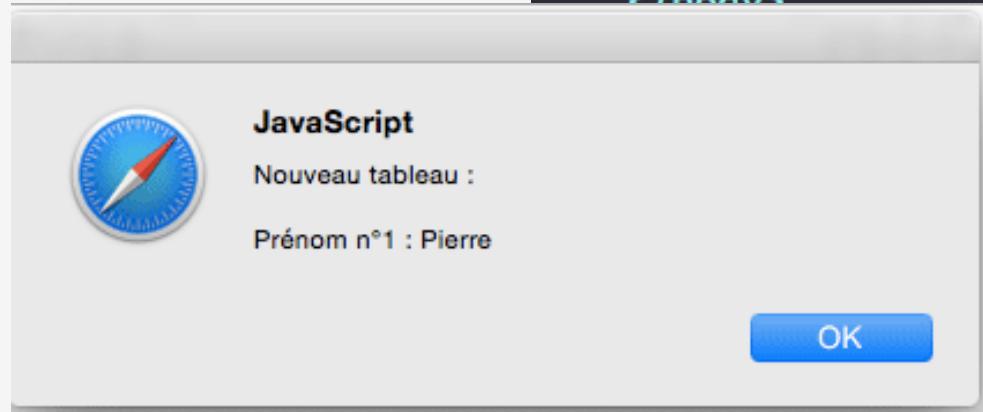


Pour supprimer, il suffit de n'indiquer **aucun** élément à ajouter à **splice()**.

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'], p = '';
 //On supprime trois éléments à partir de 'Victor'
 prenoms.splice(1, 3);

 //On récupère les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }

 //On affiche les prénoms du tableau
 alert('Nouveau tableau :\n\n' +p);
 </script>
</body>
```



# Classer les éléments

---

- La méthode **sort()** va nous permettre de trier les éléments d'un tableau dans l'ordre croissant ou alphabétique
- les valeurs commençant par des majuscules vont être classées avant celles en minuscules.
- la méthode sort() va classer les valeurs en les comparant caractère par caractère:
  - problèmes pour trier les nombres
  - « 25 » va être considéré comme supérieur à « 100 »
    - on utilisera en plus de sort() une fonction de comparaison
    - A voir par la suite

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'pierre', 'Julia', 'julia'], p = '';

 //On classe dans l'ordre alphabétique
 prenoms.sort();

 //On récupère les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }

 //On affiche les prénoms du tableau
 alert('Nouveau tableau :\n\n' +p);
 </script>
```

### JavaScript

Nouveau tableau :

Prénom n°1 : Julia  
Prénom n°2 : Pierre  
Prénom n°3 : julia  
Prénom n°4 : pierre

OK

# Inverser l'ordre

- la méthode **reverse()**
- Il est intéressant de l'utiliser avec la méthode **sort()**
  - classer des éléments dans l'ordre inverse de l'ordre alphabétique ou décroissant.



```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'pierre', 'Julia', 'julia'], p = '';
 //On classe dans l'ordre alphabétique
 prenoms.sort();

 //On inverse l'ordre
 prenoms.reverse();

 //On récupère les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }

 //On affiche les prénoms du tableau
 alert('Nouveau tableau :\n\n' +p);
 </script>
</body>
</html>
```

# La méthode join()

- La méthode **join()** va retourner les différentes valeurs d'un tableau sous forme de chaîne de caractères.
- Choisir le type de séparateur voulu entre les différentes valeurs de notre tableau (un **espace**, une **virgule**, **etc.**).
- Elle ne modifie pas notre tableau, elle renvoie seulement une chaîne de caractères créée depuis

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'pierre', 'Julia', 'julia'], p = '';

 /*join() retourne les valeurs d'un tableau sous forme de
 chaîne de caractères/
 var chaine = prenoms.join(' / ');

 //On récupère les valeurs de notre tableau
 for(var i = 0; i < prenoms.length; i++){
 p += 'Prénom n°' + (i+1) + ' : ' + prenoms[i] + '\n';
 }

 //On affiche les prénoms du tableau
 alert('Notre tableau :\n' + p +
 '\n\nLa chaîne renvoyée par join() : ' + chaine);
 </script>
</body>
</html>
```



# Créer tableau depuis un autre

- La méthode **slice()** nous permet de créer un nouveau tableau en extrayant des éléments d'un tableau de base. Elle a deux arguments :
  - une position de départ et
  - une position de fin pour couper.
- On peut ne préciser qu'un seul argument qui sera la position de départ.
  - **slice()** coupera jusqu'à la fin du tableau de départ.
- Elle n'impacte pas le tableau de départ.
  - Création d'un nouveau tableau à partir de notre tableau

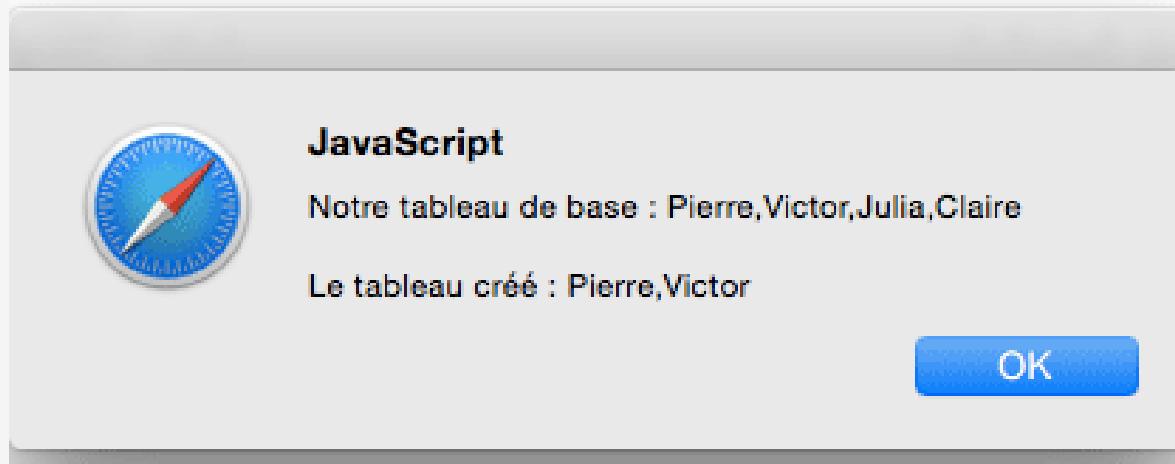
Noter que nous n'avons pas utilisé de boucle **for** pour afficher les valeurs des tableaux, mais tout simplement avec un **alert()**.

- Déconseillé car le formatage est non contrôlé

```
<body>
 <h1>Les tableaux</h1>
 <script>
 var prenoms=['Pierre', 'Victor', 'Julia', 'Claire'];

 /*slice() nous permet de créer un nouveau tableau à partir
 d'un autre*/
 var garcons = prenoms.slice(0, 2);

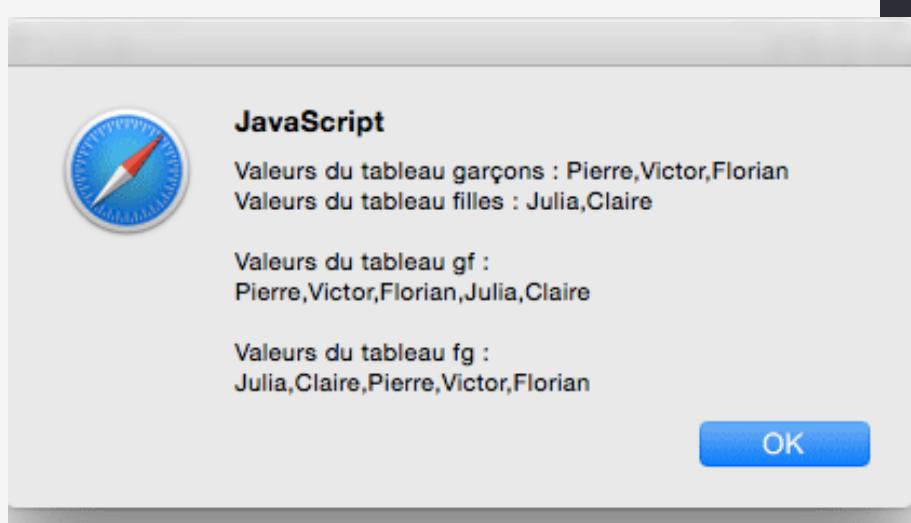
 //On affiche à la suite (sans boucle for) les valeurs des tableaux
 alert('Notre tableau de base : ' + prenoms +
 '\nLe tableau créé : ' + garcons);
```



# Concaténation de tableaux

---

- La méthode **concat()** permet de « concaténer » différents tableaux entre eux ;
- Prend en arguments les tableaux que l'on souhaite fusionner à un premier de départ pour en former un nouveau.
  - Les tableaux de base ne sont pas modifiés.
- On peut fusionner autant de tableaux que l'on veut entre eux.
- On peut aussi placer tous les autres tableaux à fusionner avec ce premier tableau en arguments de **concat()**, en les séparant par des virgules.



```
<body>
 <h1>Les tableaux</h1>
 <script>
 //On crée deux tableaux
 var garcons = ['Pierre', 'Victor', 'Florian'];
 var filles = ['Julia', 'Claire'];

 /*On fusionne les deux tableaux ci-dessus avec
 concat() pour en créer un troisième, nouveau/
 var gf = garcons.concat(filles);

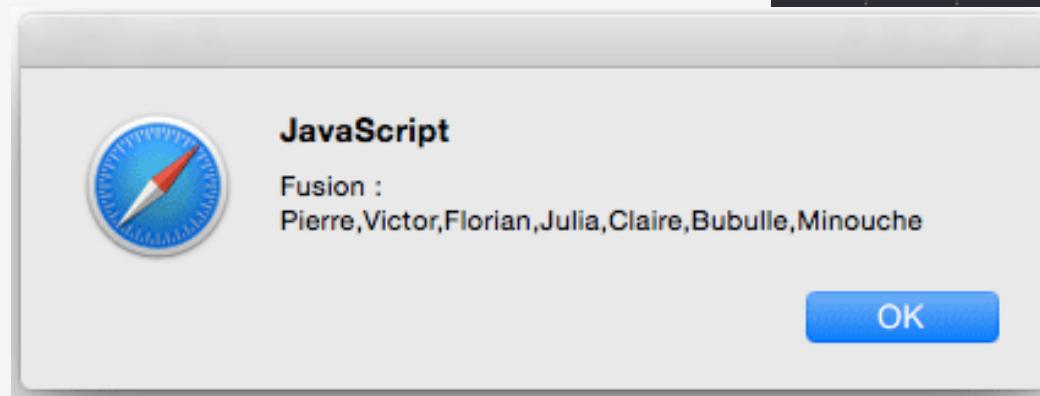
 //On aurait pu également bien sûr faire ceci :
 var fg = filles.concat(garcons);

 //On affiche les différents tableaux
 alert('Valeurs du tableau garçons : ' + garcons +
 '\nValeurs du tableau filles : ' + filles +
 '\n\nValeurs du tableau gf : ' + gf +
 '\n\nValeurs du tableau fg : ' + fg);
 </script>
</body>
</html>
```

```
<body>
 <h1>Les tableaux</h1>
 <script>
 //On crée deux tableaux
 var garcons = ['Pierre', 'Victor', 'Florian'];
 var filles = ['Julia', 'Claire'];
 var animaux = ['Bubulle', 'Minouche'];

 //On crée un nouveau tableau à partir de nos 3 tableaux
 var fusion = garcons.concat(filles, animaux);

 //On affiche le tableau créé
 alert('Fusion : ' + fusion);
 </script>
```



# Exercice de récapitulation

Le but est de récupérer le login et mot de passe afin de passer une page d'accueil.

1. Il faut demander à l'utilisateur son **login**. S'il choisit annuler, on affiche la page blanche.
  2. Ensuite, on compare le login entré avec celui attendu
    - Login est déjà déclaré dans une variable avant
    - S'il est différent, on redemande le login, sinon, on continue le traitement.
  3. On demande ensuite le **password**.
    - Password est déjà déclaré dans une variable avant
    - On vérifie qu'il est correct et on continue. S'il ne l'est pas, on redemande 2 fois ce password.
    - S'il est toujours incorrect au 3<sup>ème</sup> essai, on affiche un message d'erreur à l'écran, et on fini.
  4. Enfin, on affiche l'écran d'accueil.
- 
- Pour afficher à l'écran, utiliser la méthode **document.write()**, en passant la chaîne à afficher en paramètre.
  - Cet exercice utilisera vos connaissances au niveau des **fonctions**, des **variables**, des **boucle**, des **conditions**, et bien entendu des **boites de dialogue**.
  - Vous pouvez aussi stocker les données d'accès (login et password) dans un objet **Array**.

# Programme de formation

I- Les technologies du Web

II- Le langage JavaScript, prise en mains

III- Evénements et données

IV- Gestion de formulaires HTML

V- Interaction avec les CSS

VI- Manipulation du DOM XML

VII- Ajax

## III- Evénements et données

1. Organisation des événements. Impact des événements sur les types de navigateurs et versions de DOM.
2. Positionner des écouteurs sur des événements par programme et paramétrage de balises HTML.
3. Règles pour faire un codage multinavigateur.
4. Créer, détruire des écouteurs.
5. Les traitements événementiels JavaScript : gestionnaire clavier, souris, formulaires, rollover, menus dynamiques.
6. L'objet Event et son utilisation.
7. Les objets du DOM (window, document...) et leur manipulation.
8. Manipulation des URL (redirections http...).
9. Gestion des cookies (lecture et écriture).

*Exercices: Programmation d'événements*

## 2.1- Organisation des événements. Impact des événements sur les types de navigateurs et versions de DOM

Ce que nous cherchons à faire ici, c'est se familiarise avec les évènements présents dans le navigateur et que JavaScript peut « **capturer** » et nous offrir., puis certainement, nous chercherons à réaliser des « **réactions** » adéquates selon nos gouts et besoins.

Nous allons pouvoir:

1. Définir les évènement
2. Déclencher un évènement
  - Automatiquement par le navigateur
  - Après une manipulation de notre utilisateur
3. Connaitre les évènements les plus courants
  - onclick , onmouseover , onmouseout , onkeydown , etc.
4. Apprendre à réagir aux évènements via le DOM

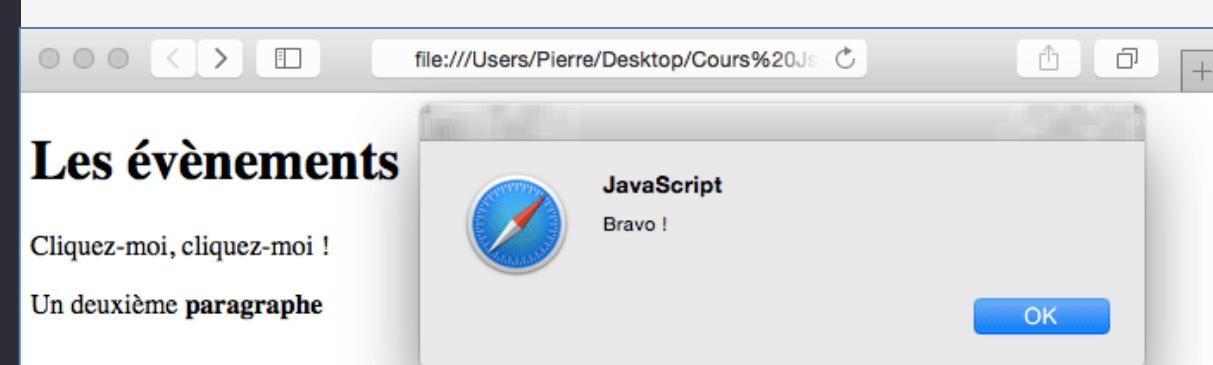
# 1er exemple

- Afficher une boîte de dialogue de type **alert()** lorsqu'un utilisateur clique sur un paragraphe dans notre page web.
- On utilise l'attribut **onclick** HTML sur notre paragraphe puis on va créer notre **alert()** en JavaScript.
- HTML nous autorise à placer du code **JavaScript** directement en *valeur* de l'attribut **onclick**.

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

 <body>
 <h1 id="gros_titre">Les évènements</h1>

 <p onclick="alert('Bravo !');">Cliquez-moi, cliquez-moi !</p>
 <p>Un deuxième paragraphe</p>
 </body>
</html>
```



# L'utilisation du mot clef **this**

- **this** nous sert de référence à différents objets selon le contexte.
- **this** va faire référence à l'objet (représentant l'élément HTML) qui est le sujet de l'événement.
- En utilisant **this** dans notre code JavaScript dans ce cas, on va donc travailler sur l'élément **p** en soi.

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

 <body>
 <h1 id="gros_titre">Les évènements</h1>

 <p onclick="this.textContent='Merci !';">Cliquez-moi, cliquez-moi !</p>
 <p>Un deuxième paragraphe</p>
 </body>
</html>
```

## Les évènements

Cliquez-moi, cliquez-moi !

Un deuxième paragraphe

## Les évènements

Merci !

Un deuxième paragraphe

# Les évènements et le DOM

- Le DOM HTML permet d'assigner des gestionnaires d'évènements spécifiques à des éléments HTML en utilisant **JS**
- Nous n'allons donc plus utiliser des attributs HTML mais du **pure** code JavaScript

On a 2 possibilités avec le DOM:

1. utiliser des **propriétés** qui vont assigner un gestionnaire d'événement à un élément spécifique en HTML,
  - ancienne méthode et vous ne devriez plus l'utiliser aujourd'hui car elle possède des limitations
  - on ne peut pas assigner plusieurs fois le même gestionnaire d'événement à un élément HTML
2. utiliser la méthode **addEventListener()**

# 1-Usage des propriétés

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

 <body>
 <h1 id="gros_titre">Les évènements</h1>

 <p>Cliquez-moi, cliquez-moi !</p>
 <p>Un deuxième paragraphe</p>
 <script>
 //On accède à notre premier paragraphe
 var p1 = document.querySelector('p');

 /*Création d'un gestionnaire d'évènements pour
 l'évènement "onclick"/
 p1.onclick = function(){
 this.innerHTML = 'Bravo !';
 this.style.color = 'orange';
 };
 </script>
 </body>
</html>
```



# Exécution des fonctions

- Noter l'usage des fonctions **anonymes**
- on commence par accéder à notre premier paragraphe puis on lui attache un évènement de type **onclick**
- Ici, nous utilisons une fonction anonyme afin que celle-ci **ne s'exécute pas immédiatement**.
- UPDATE: Donc, pour exécuter une fonction anonyme, on peut:
  1. la transformer en fonction auto invoquée,
  2. l'enfermer dans une variable puis appeler cette variable avec un couple de parenthèses
  3. **l'assigner à un évènement:** c'est notre exemple précédent
- **Alternative: fonction avec nom**

```

<!DOCTYPE html>
<html>
 <head>
 <title>Les évènements</title>
 <meta charset="utf-8">
 </head>

 <body>
 <h1 id="gros_titre">Les évènements</h1>

 <p>Cliquez-moi, cliquez-moi !</p>
 <p>Un deuxième paragraphe</p>
 <script>
 //On accède à notre premier paragraphe
 var p1 = document.querySelector('p');

 /*Création d'un gestionnaire d'évènements pour
 * l'évènement "onclick"*/
 p1.onclick = bravo;

 function bravo(){
 this.innerHTML = 'Bravo !';
 this.style.color = 'orange';
 };
 </script>
 </body>
</html>

```

# Méthode `addEventListener()`

## 2.2- Positionner des écouteurs sur des événements par programme et paramétrage de balises HTML

La Méthode **addEventListener()** va nous permettre de lier du code à un évènement.

- On parlera alors de gestionnaire d'évènements
- Le code sera alors exécuté dès le déclenchement de l'évènement
- Appartient à l'objet **Element**

Cette méthode a besoin de 2 arguments :

1. le **nom** de l'évènement déclencheur de l'action
2. le **code** relatif à l'action à effectuer

Les événements vont avoir des noms similaires aux attributs HTML mais sans le « on »

- par exemple, **onclick** devient **click**

## 1er exp.:

Event Listener:

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

 <body>
 <h1 id="gros_titre">Les évènements</h1>

 <p>Cliquez-moi, cliquez-moi !</p>
 <p>Un deuxième paragraphe</p>
 <script>
 //On accède à notre premier paragraphe
 var p1 = document.querySelector('p');

 //On accroche un gestionnaire d'évènements à p1
 p1.addEventListener('click',changeTexte);

 /*On construit notre fonction changeTexte qui ne sera
 exécutée que lors du déclenchement de l'évènement/
 function changeTexte(){
 this.innerHTML = 'Bravo !';
 this.style.color = 'orange';
 };
 </script>
 </body>
</html>
```

# Attention aux parenthèses!

On ne précise pas l'habituel couple de parenthèses après notre fonction

- Afin que celle-ci ne s'exécute pas immédiatement !
- mais seulement après le déclenchement de l'évènement (le clic sur notre paragraphe)

```
// On accroche un gestionnaire d'évènements à p1
p1.addEventListener('click',changeTexte);
```

On passe

- un nom d'évènement en premier argument de la méthode addEventListener()
- puis le nom d'une fonction à exécuter en second argument

Elle peut être une fonction **imbriquée anonyme**

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

 <body>
 <h1 id="gros_titre">Les évènements</h1>

 <p>Cliquez-moi, cliquez-moi !</p>
 <p>Un deuxième paragraphe</p>
 </body>
 <script>
 //On accède à notre premier paragraphe
 var p1 = document.querySelector('p');

 //On accroche un gestionnaire d'évènements à p1
 p1.addEventListener('click',function(){
 this.innerHTML = 'Bravo !';
 this.style.color = 'orange';
 });

 </script>
</html>
```

# Pourquoi utiliser addEventListener() ?

L'un des grands avantages de la méthode addEventListener() est:

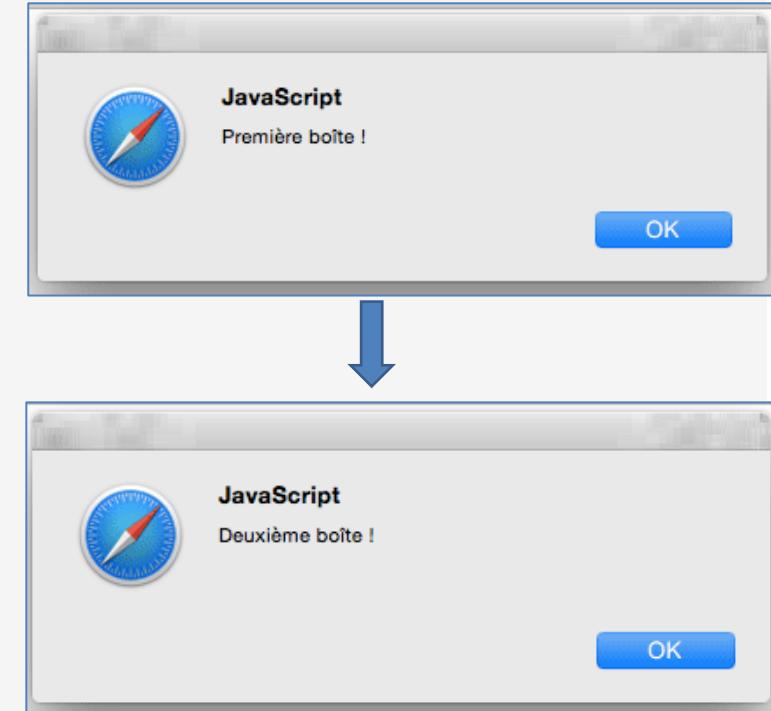
- pouvoir lier **plusieurs gestionnaires** d'évènements de même type sur un élément HTML
- Par exemple: afficher différents messages suite à un clic

```
<script>
 //On accède à notre premier paragraphe
 var p1 = document.querySelector('p');

 //On accroche deux gestionnaires d'évènements à p1
 p1.addEventListener('click',Message1);
 p1.addEventListener('click',Message2);

 //Création des fonctions
 function Message1(){
 alert('Première boîte !');
 };

 function Message2(){
 alert('Deuxième boîte !');
 };
</script>
</html>
```



# Lier des évènements différents

## Changement si l'utilisateur

- déplace le curseur de sa souris sur un élément (**mouseover**)
- et un autre code lorsqu'il reste cliqué dessus (**mousedown**)

```
<script>
 //On accède à notre premier paragraphe
 var p1 = document.querySelector('p');

 //On accroche deux gestionnaires d'évènements à p1
 p1.addEventListener('mouseover',Fonction1);
 p1.addEventListener('mousedown',Fonction2);

 //Création des fonctions
 function Fonction1(){
 this.innerHTML = 'Cliquez moi maintenant !';
 this.style.backgroundColor = 'orange';
 };

 function Fonction2(){
 this.innerHTML = 'Bravo !';
 this.style.color = '#26C';
 this.style.fontWeight = 'bold';
 this.style.fontSize= '24px';
 };
</script>
```

## Les évènements

Cliquez moi maintenant !

Un deuxième paragraphe

## Les évènements

Bravo !

Un deuxième paragraphe

## 2.3- Créer, détruire des écouteurs

Voir exercice de codage: Solution pour la gestion des transaction de revenus et de dépenses

[/INETUM JavaScript](#) (partie projet)

```
function Transaction(name, amount)
{
 this.name = name;
 this.amount = amount;

 this.getName = function () {
 return this.name;
 };

 this.getAmount = function () {
 return this.amount;
 };

 this.getInfo = function () {
 return "Transaction" +
 this.name + " de
 montant " + this.amount;
 };
}
```

## 2.4- Les traitements événementiels JavaScript : gestionnaire clavier, souris, formulaires, rollover, menus dynamiques

Exemple à 4 EventListeners

**mouseout:** l'utilisateur déplace le curseur de sa souris en dehors d'un élément

**Mouseup:** l'utilisateur relâche le clic

```
function Fonction1(){
 this.innerHTML = 'Cliquez moi maintenant !';
 this.style.backgroundColor = 'orange';
};
```

```
function Reset1(){
 this.innerHTML = 'Passez sur moi svp';
 this.style.backgroundColor = '';
};
```

```
var p1 = document.querySelector('p');

p1.addEventListener('mouseover',Fonction1);
p1.addEventListener('mouseout',Reset1);
p1.addEventListener('mousedown',Fonction2);
p1.addEventListener('mouseup',Reset2);
```

```
function Fonction2(){
 this.innerHTML = 'Bravo !';
 this.style.color = '#26C';
 this.style.fontWeight = 'bold';
 this.style.fontSize= '24px';
};
```

```
function Reset2(){
 this.innerHTML = 'Passez sur moi svp';
 this.style.color = '';
 this.style.fontWeight = '';
 this.style.fontSize= '';
};
```

# QUIZ

---

Pour créer un gestionnaire d'évènements, il est généralement recommandé d'utiliser:

- A. Les attributs HTML relatifs aux évènements
- B. Les propriétés JavaScript relatives aux évènements
- C. La méthode addEventListener()

Citez les deux phases de propagation des évènements en JavaScript.

? A quoi correspondent-elles ?

# Propagation des évènements

Présentation et explication

Comprendre les 2 phases de propagation

Gérer l'ordre de déclenchement des évènements

## 2.6- L'objet Event et son utilisation; Propagation

Caractériser l'ordre dans lequel différents évènements vont se déclencher

Imaginons une page web comportant

- un **div** et un élément **p** à l'intérieur de ce div
- 2 gestionnaires d'évènements **click** à notre **div** et à notre **p**

lorsqu'un utilisateur va cliquer sur notre paragraphe

- les deux évènements vont se déclencher
- puisque notre paragraphe est contenu dans notre div
- **Dans quel ordre** vont se déclencher les évènements ?

```

<!DOCTYPE html>
<html>
 <head>
 <title>Les évènements</title>
 <meta charset="utf-8">
 </head>

 <body>
 <h1 id="gros_titre">Les évènements</h1>

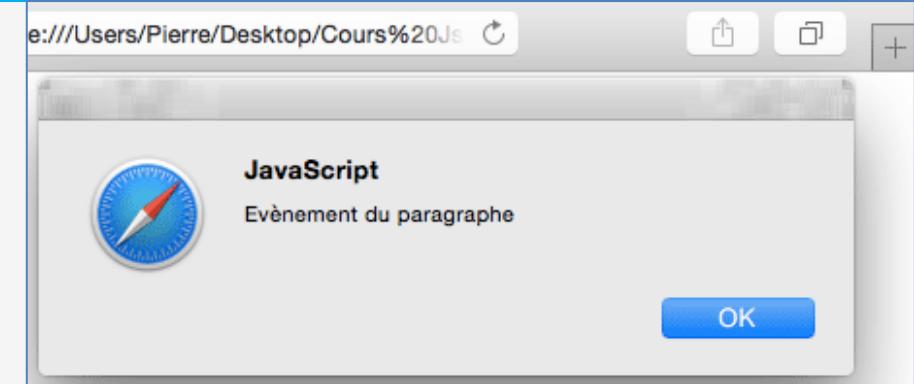
 <div id="div">
 <p id="p">Un premier paragraphe</p>
 <p>Un deuxième paragraphe</p>
 </div>
 </body>
 <script>
 //On accède à notre div et à premier paragraphe
 var div = document.getElementById('div');
 var p1 = document.getElementById('p');

 //On crée deux gestionnaires d'évènements pour click
 div.addEventListener('click',MessageDiv);
 p1.addEventListener('click',MessageP);

 function MessageDiv(){
 alert('Evènement du div');
 };

 function MessageP(){
 alert('Evènement du paragraphe');
 };
 </script>
</html>

```



La position l'influence pas l'ordre des évènement

- La propagation part toujours depuis la racine DOM vers l'élément le plus profond (**phase de capture**)
- Remontée (**phase de bouillonnement**)

Les évènements se déclenchent par défaut dans la phase de remontée (de bouillonnement)

On peut spécifier la phase **et la changer** par un 3ème argument optionnel de addEventListner() de valeur:

- **True**: déclencher l'évènement lors de la phase de capture
- **false** (valeur par défaut): ne déclencher l'évènement que lors de la phase de bouillonnement

```

<head>
 <title>Les évènements</title>
 <meta charset="utf-8">
 <style>
 div{
 border: 1px solid orange;
 background-color: #29B;
 padding: 20px;
 margin: 10px;
 }
 </style>
</head>

<body>
 <h1 id="gros_titre">Les évènements</h1>

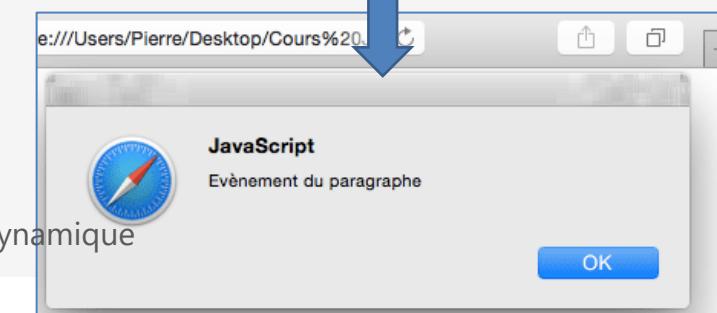
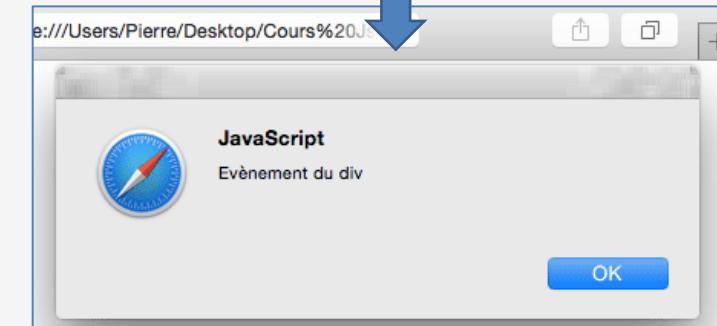
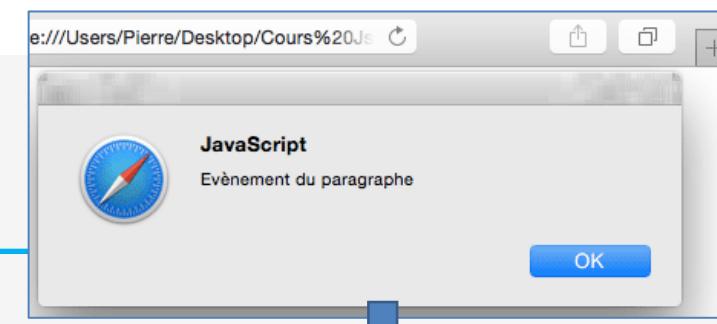
 <div id="div1">
 <p id="p1">Phase de bouillonnement utilisée</p>
 </div>
 <div id="div2">
 <p id="p2">Phase de capture utilisée</p>
 </div>

 <script>
 //On accède à div1, p1, div2 et p2
 var div1 = document.getElementById('div1');
 var p1 = document.getElementById('p1');
 var div2 = document.getElementById('div2');
 var p2 = document.getElementById('p2');

 //Phase de bouillonnement utilisée pour div1 et p1
 div1.addEventListener('click',MessageDiv1);
 p1.addEventListener('click',MessageP1);
 function MessageDiv1(){alert('Evènement du div');}
 function MessageP1(){alert('Evènement du paragraphe');}

 //Phase de capture utilisée pour div2 et p2
 div2.addEventListener('click',MessageDiv2,true);
 p2.addEventListener('click',MessageP2,true);
 function MessageDiv2(){alert('Evènement du div');}
 function MessageP2(){alert('Evènement du paragraphe');}
 </script>

```



# Mixes de bouillonnement/capture

1. P1
2. Strong
3. Div
4. body

```
<body>
 <h1 id="gros_titre">Les évènements</h1>
 <div id="div1">
 <p id="p1">Un exemple plus complexe</p>
 </div>

 <script>
 //On accède à body, div1, p1 et à strong
 var body = document.body;
 var div1 = document.getElementById('div1');
 var p1 = document.getElementById('p1');
 var strong = document.querySelector('strong');

 //Phase de capture utilisée pour strong et p1
 p1.addEventListener('click',CP1,true);
 strong.addEventListener('click',CS,true);
 function CP1(){alert('Capture paragraphe');}
 function CS(){alert('Capture strong');}

 //Phase de bouillonnement utilisée pour div1 et body
 div1.addEventListener('click',BD1);
 body.addEventListener('click',BB);
 function BD1(){alert('Bouillonnement div');}
 function BB(){alert('Bouillonnement body');}
 </script>
</body>
</html>
```

# L'objet Event

Possède des propriétés et des méthodes

- informant sur le contexte de l'évènement déclenché
- impactant l'environnement

N'est accessible **que durant le déclenchement** d'un évènement

- Accès au sein de la fonction servant à exécuter une action lors du déclenchement de l'évènement

Possède une dizaine de propriétés. Nous allons voir

- target,
- currentTarget
- type

# Propriétés target & currentTarget

La propriété **target** retourne le type de l'élément qui a déclenché l'évènement

La propriété **currentTarget** retourne le type de l'élément portant le gestionnaire de l'évènement déclenché

Ces deux propriétés sont très utiles pour connaître

- quel élément possède le gestionnaire d'évènement
- dans quelle phase (capture ou bouillonnement) il s'est déclenché

```
<body>
 <h1 id="gros_titre">Les évènements</h1>
 <div id="div1">
 <p id="p1">Qui est le déclencheur ? Le porteur ?</p>
 </div>

 <script>
 //On accède à div1 et p1
 var div1 = document.getElementById('div1');
 var p1 = document.getElementById('p1');

 //On demande à div1 de réagir en cas de clic
 div1.addEventListener('click', Message);

 //On utilise target et currentTarget durant l'évènement
 function Message(event){
 this.innerHTML = 'Element déclencheur : ' + event.target +
 '
Element portant l\'évènement : ' + event.currentTarget;
 }
 </script>
</body>
</html>
```

# Les URL sous JavaScript

Il est possible d'exprimer une URL absolue comme <https://monsite.fr/chemin/page.html> ou des URL relatives comme [chemin/page.html](#).

```
// Similar as an HTTP redirect
window.location.replace("https://my-website.fr/");
// Similar as clicking on a link
window.location.href = "https://my-website.fr/";
```

## Récupérer l'URL

nous pouvons utiliser l'objet **window.location** proposé par défaut dans JavaScript

```
var url = window.location.protocol + "//" + window.location.host + window.location.pathname;
```

## Récupérer le protocole (HTTP, HTTPS, FILE...)

si ce code est exécuté sur une page ayant l'URL <https://monsite.fr>, alors la variable **protocole** aura la valeur « [https](https://monsite.fr) ».

```
var protocol = window.location.protocol;
```

## Récupérer le nom de domaine

Si ce code est exécuté sur une page ayant l'URL <https://monsite.fr/chemin/page.html>, alors la variable **host** aura la valeur « [monsite.fr](https://monsite.fr) »

```
var host = window.location.host;
```

## Récupérer le chemin de l'URL

Si ce code est exécuté sur une page ayant l'URL <https://monsite.fr/chemin/page.html>, alors la variable **path** aura la valeur « `chemin/page.html` »

```
var path = window.location.pathname;
```

Vous pouvez aussi récupérer chaque éléments du chemin dans un tableau avec le code

```
var pathArray = window.location.pathname.split("/");
```

## Récupérer l'ancre

si ce code est exécuté sur une page ayant l'URL <https://monsite.fr/chemin/page.html#mon-ancre>, alors la variable **hash** aura la valeur « `#mon-ancre` ».

```
var path = window.location.hash;
```

Vous pouvez retirer simplement le dièse (« # ») en utilisant **window.location.hash.substr(1)**

# Les Cookies sous JavaScript

Ce sont de petits fichiers textes enregistrés sur l'ordinateur du visiteur afin de

- retenir des informations sur l'utilisateur, par exemple:
  - pseudo de l'utilisateur par exemple
  - Ses préférences pour un produit à acheter par exemple
- Chaque cookie est destiné à une information unique
  - Utiliser 2 cookies si on besoin de 2 variables , par exp. '[pseudonyme](#)' et '[date de naissance](#)' du visiteur
  - On peut les supprimer depuis le navigateur

## Outils / Options / Vie privée

- Pour cela, on va utiliser le descripteur d'accesseur **document.cookie**
  - Possède fonctions (getter/setter) permettant l'accès
- La portée des cookies : chemin (répertoire) est accessible avec l'option **path**.
  - Le chemin fourni doit être absolu. Par défaut, un cookie est accessible dans la page courante.

## [path =/coursDHL](#)

```
document.cookie = 'user=Bassem';
//Crée ou met à jour un cookie 'user'
alert(document.cookie);
//Affiche la liste des cookies
```

# Propriété? Dans la chaîne de définition du Cookie

- L'option **expires** permet de préciser une date d'expiration pour un cookie
- option **max-age** pour définir la date d'expiration d'un cookie en secondes à partir du moment actuel.
- L'option **secure** permet d'indiquer qu'un cookie doit être envoyé uniquement via HTTPS et ne pas l'être via HTTP  
→ très utile
- L'option **samesite** empêche le navigateur d'envoyer un cookie lors d'une requête cross-site
- L'option **httpOnly** ne dépend pas du JavaScript mais va avoir un effet important sur l'utilisation des cookies

```
let date = new Date(Date.now() + 86400000); //8640000ms = 1 jour
date = date.toUTCString();

//Crée ou met à jour un cookie 'user'
document.cookie = 'user=FormationDHL; path=/; expires=' + date;

//Supprime le cookie en lui passant une date d'expiration passée
document.cookie = 'user=; path=/; expires=Thu, 01 Jan 1970 00:00:00 UTC';
```

# Exemples de fonctions utilisables pour cookies

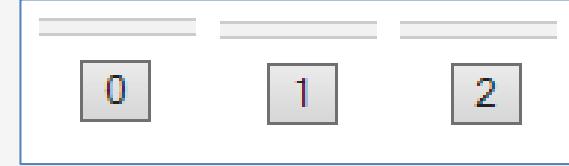
```
function setCookie(cname, cvalue, exdays) {
 var d = new Date();
 d.setTime(d.getTime() + (exdays*24*60*60*1000));
 var expires = "expires="+ d.toUTCString();
 document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
```

[DEMO](#)

```
function getCookie(cname) {
 var name = cname + "=";
 var decodedCookie = decodeURIComponent(document.cookie);
 var ca = decodedCookie.split(';');
 for(var i = 0; i <ca.length; i++) {
 var c = ca[i];
 while (c.charAt(0) == ' ') {
 c = c.substring(1);
 }
 if (c.indexOf(name) == 0) {
 return c.substring(name.length, c.length);
 }
 }
 return "";
}
```

# Exercices d'application

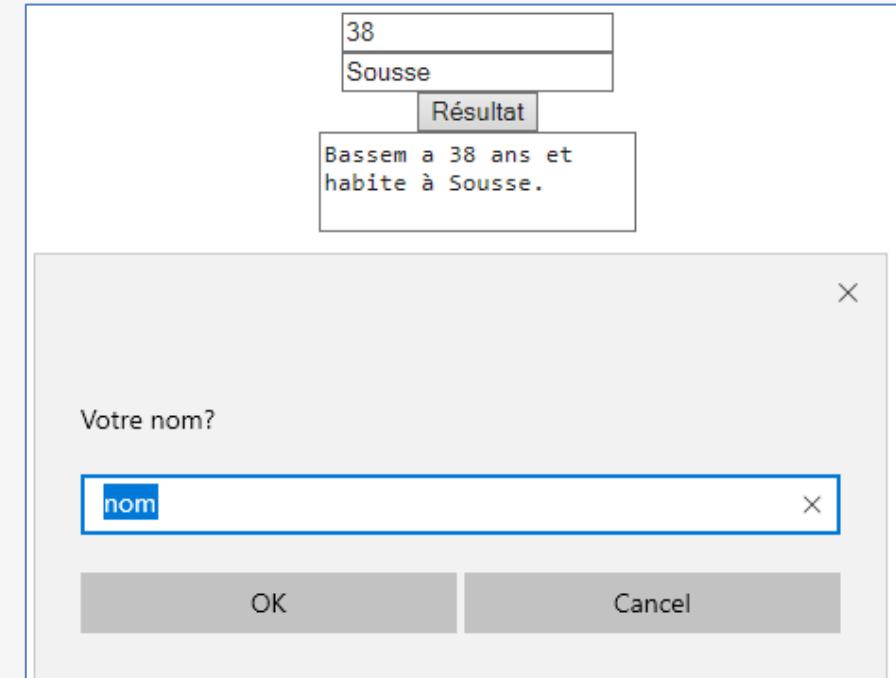
**Exercice 1:** L'exercice consiste à faire un bouton dont la valeur s'incrémente à chaque clic. Ce n'est pas très compliqué.



**Exercice 2:** L'exercice consiste à demander 3 informations à un utilisateur, puis de les afficher dans une zone de texte en faisant un phrase.

Il faudra demander le nom à l'aide d'une boîte de message, ainsi que l'âge et la ville avec des lignes de texte.

Ensuite, en cliquant sur un bouton, on affiche une phrase contenant les 3 informations dans une zone de texte.



# Programme de formation

I- Les technologies du Web

II- Le langage JavaScript, prise en mains

III- Evénements et données

IV- Gestion de formulaires HTML

V- Interaction avec les CSS

VI- Manipulation du DOM XML

VII- Ajax

## IV- Gestion de formulaires HTML

1. Manipulation de contenu de formulaires.
2. Accès et modification dynamique des composants du formulaire : zone de saisie, cases à cocher, cases d'options...
3. Fonctions de validation de formulaire.
4. Evénements liés aux éléments de formulaire : changement, initialisation, clic, etc.
5. Les expressions régulières RegEx sous JavaScript

**Exercices:** *Fonctions personnalisées contrôlant les activités de l'utilisateur.*

# Rappels

Les formulaires servent à recevoir des données envoyées par les utilisateurs

- Par exemple: nom, prénom, adresse, etc.

les formulaires sont

- écrits en HTML,
- mis en forme en CSS
- les informations envoyées sont
  - analysées par PHP
  - stockées sous MySQL

```
<form method="post" action="traitement.php">
 ...
</form>
```

Rappel: Sous HTML: Elément **form** avec les attributs

- **Method:** La méthode d'envoi des données (post ou get)
- **Action:** la page PHP qui va assurer le traitement

# Formulaire simple en HTML

Un formulaire très simple en HTML, demandant à l'utilisateur

- son prénom,
- numéro de téléphone
- et adresse mail

```
<form method="post" action="traitement.php">
 <label for='prenom'>Entrez votre prénom svp : </label>
 <input type='text' name='prenom' id='prenom'>

 <label for='mail'>Entrez votre mail : </label>
 <input type='email' name='mail' id='mail'>

 <label for='tel'>Numéro de téléphone :</label>
 <input type='tel' name='tel' id='tel'>

 <input type='submit' value='Valider'>
</form>
```



# Données utilisateurs problématiques

Rien n'empêche les utilisateurs de:

- envoyer le formulaire sans avoir rempli certains champs
- donner des informations erronées
  - Texte à place d'un numéro de téléphone par exemple

→ Toujours vérifier la structure des données envoyées

Nous allons le « forcer » à

- Renseigner quelque chose
- Suivre la forme d'un numéro de téléphone (8 chiffres consécutifs)

# Première validation en HTML

HTML possède déjà des attributs nous permettant de placer nos premières contraintes

- attribut **required**: rend obligatoire le remplissage d'un champ
- attributs **minlength** et **maxlength**: définir la taille d'un champ input type="text"
- attributs **min** et **max**: pour les input type="number" et "date"

```
<form method="post" action="traitement.php">
 <label for='prenom'>Entrez votre prénom svp : </label>
 <input type='text' name='prenom' id='prenom' maxlength='20' required>

 <label for='mail'>Entrez votre mail : </label>
 <input type='email' name='mail' id='mail' required>

 <label for='tel'>Numéro de téléphone :</label>
 <input type='tel' name='tel' id='tel' required>

 <input type='submit' value='Valider'>
</form>
```

```
<form method="post" action="traitement.php">
 <label for='prenom'>Entrez votre prénom svp : </label>
 <input type='text' name='prenom' id='prenom' maxlength='20' required>

 <label for='mail'>Entrez votre mail : </label>
 <input type='email' name='mail' id='mail' required>

 <label for='tel'>Numéro de téléphone :</label>
 <input type='tel' name='tel' id='tel' required pattern="^((\+\d{1,3}(-|\?
)?\d\)?(-|)?\d{1,5})|(\d{2,6}))(-|)?(\d{3,4})(-|)?(\d{4})((ext)\d{1,5}){0,1}$">

 <label for='the_search'>Recherche :</label>
 <input type="search" placeholder="Recherche" name="the_search" required>

 <label for='anniversaire'>Date :</label>
 <input type="date" name="anniversaire" required>

 <label for='holidays'>Mois :</label>
 <input type="month" name="holidays" required>

 <label for='howmuch'>Nombre :</label>
 <input type="number" name="howmuch" required>

 <label for='Slider'>Slider :</label>
 <input type="range" name="Slider" required>

 <label for='textcolor'>Couleur :</label>
 <input type="color" value="#fad345" name="textcolor" required>

 <input type='submit' value='Valider'>
</form>
```

# Validation en JavaScript

Ce langage offre une vérification du côté client

- des propriétés spécialement créées pour la validation d'éléments de formulaire
- également d'utiliser nos regex

Cela ne nous dispense pas d'effectuer également des vérifications de l'intégrité des données côté serveur (Php)

- L'utilisateur peut désactiver JavaScript sur son navigateur

JavaScript sera

- Utile pour les user bienveillants
- Contournable pour ceux malveillants
  - Attaques toujours possibles

# Exemple de validation en JS

Pour le formulaire précédent, nous voulons vérifier:

- Le champ n'est pas vide ;
- Le champ ne contient que des lettres (accentuées ou pas), des tirets et des espaces (pour gérer les prénoms composés) ;
- Le champ contient entre 2 et 30 caractères;
- Aussi
  - afficher un message d'erreur lorsque le champ n'est pas (bien) rempli
  - avec une indication sur l'erreur rencontrée

file:///Users/Pierre/Desktop/Cours%20Js

# Les formulaires HTML

Entrez votre prénom svp :

Entrez votre mail :

Numéro de téléphone :

Prénom manquant

```
<body>
 <h1>Les formulaires HTML</h1>

 <form method="post" action="traitement.php">
 <label for='prenom'>Entrez votre prénom svp : </label>
 <input type='text' name='prenom' id='prenom' maxlength='20' required>

 <label for='mail'>Entrez votre mail : </label>
 <input type='email' name='mail' id='mail' required>

 <label for='tel'>Numéro de téléphone :</label>
 <input type='tel' name='tel' id='tel' required>

 <input type='submit' value='Valider' id='bouton_envoi'>
 </form>

 <script>
 var formValid = document.getElementById('bouton_envoi');

 formValid.addEventListener('click', validation);

 function validation(){
 ...
 }
 </script>
</body>
</html>
```

M2ii-INEUUM - JavaScript et HTML dynamique

# Event. preventDefault()

La méthode **preventDefault()** annule l'événement s'il est annulable, ce qui signifie que l'action par défaut qui appartient à l'événement ne se produira pas

Par exemple, cela peut être utile lorsque:

- En cliquant sur un bouton "Envoyer", l'empêcher de soumettre un formulaire
- En cliquant sur un lien, empêchez le lien de suivre l'URL

```
<body>
 <h1>Les formulaires HTML</h1>

 <form method="post" action="traitement.php">

 <label for='prenom'>Entrez votre prénom svp : </label>
 <input type='text' name='prenom' id='prenom' maxlength='20' required>

 <label for='mail'>Entrez votre mail : </label>
 <input type='email' name='mail' id='mail' required>

 <label for='tel'>Numéro de téléphone :</label>
 <input type='tel' name='tel' id='tel' required>

 <input type='submit' value='Valider' id='bouton_envoi'>
 </form>

 <script>
 var formValid = document.getElementById('bouton_envoi');
 var prenom = document.getElementById('prenom');
 var missPrenom = document.getElementById('missPrenom');

 formValid.addEventListener('click', validation);

 function validation(event){
 //Si le champ est vide
 if (prenom.validity.valueMissing){
 event.preventDefault();
 missPrenom.textContent = 'Prénom manquant';
 missPrenom.style.color = 'red';
 }
 }
 </script>
</body>
</html>
```

# Qualité données envoyées

Nous n'allons autoriser que les lettres, les apostrophes, les tirets et les espaces pour ce champ

De plus, nous n'autoriserons l'utilisation de majuscule qu'en début de mot et nous n'autoriserons pas les tirets en début ou en fin de chaîne

```
<body>
 <h1>Les formulaires HTML</h1>
 <form method="post" action="traitement.php">
 <label for='prenom'>Entrez votre prénom svp : </label>
 <input type='text' name='prenom' id='prenom' maxlength='20' required>

 <label for='mail'>Entrez votre mail : </label>
 <input type='email' name='mail' id='mail' required>

 <label for='tel'>Numéro de téléphone :</label>
 <input type='tel' name='tel' id='tel' required>

 <input type='submit' value='Valider' id='bouton_envoi'>
 </form>

 <script>
 var formValid = document.getElementById('bouton_envoi');
 var prenom = document.getElementById('prenom');
 var missPrenom = document.getElementById('missPrenom');
 var prenomValid = /^[a-zA-ZéèïïÉÈÏÏ][a-zéèëàçïï]+([-'\s][a-zA-ZéèïïÉÈÏÏ][a-zéèëàçïï]+)?$/;

 formValid.addEventListener('click', validation);

 function validation(event){
 //Si le champ est vide
 if (prenom.validity.valueMissing){
 event.preventDefault();
 missPrenom.textContent = 'Prénom manquant';
 missPrenom.style.color = 'red';
 }
 //Si le format de données est incorrect
 else if (prenomValid.test(prenom.value) == false){
 event.preventDefault();
 missPrenom.textContent = 'Format incorrect';
 missPrenom.style.color = 'orange';
 }
 else{
 }
 }
 </script>
</body>
</html>
```

```
/^ [a-zA-ZéèïïÉÈÏÏ] [a-zéèêàçïï]+ ([-'\s] [a-zA-ZéèïïÉÈÏÏ] [a-zéèêàçïï]+)? $/;
```

## Partie 1:

- Nous demandons à l'utilisateur de commencer soit par une lettre non accentuée en majuscule ou en minuscule, soit par l'un des caractères suivants : « éèïïÉÈÏÏ ».
- cette première lettre est soit suivie par au moins une autre lettre en minuscule ou par l'un des caractères suivants : « éèêàçïï ».
- Notez l'utilisation du signe + pour demander « au moins une autre... »

## Partie 2:

- concerne le second prénom (dans le cas d'un prénom composé par exemple, ou d'un prénom contenant un apostrophe)
- doit être facultatif. Nous avons donc utilisé le signe ? sur toute cette partie en l'entourant au préalable par des parenthèses pour faire porter le point d'interrogation sur toute la partie de la regex.
- commence soit par un apostrophe, un tiret ou un espace \s, puis se poursuit par le deuxième prénom en soi

### 3.1- Organisation des événements. Impact des événements sur les types de navigateurs et versions de DOM.

Voir exercice de codage: Solution pour la gestion des transaction de revenus et de dépenses

**/INETUM JavaScript (partie projet)**

# LES EXPRESSIONS REGULIERES

Fonction **REGEX** sous JavaScript

# Introduction

---

Les expressions régulières, ou « **regex** » , forment un langage à part qui:

- n'appartient pas exclusivement au JavaScript
- ne constitue pas non plus un langage de programmation non plus
- Il est implémenté et mis en œuvre par plusieurs langages de programmation (PHP, Java, Javascript, etc.)

## Une expression régulière

- consiste en **une suite de caractères** qui forment, ensemble, un « pattern » de recherche
- nous permet de vérifier la présence de certains caractères ou suites de caractères dans une chaîne de caractères

# Usages

Les **regex** vont nous permettre par exemple de

- vérifier qu'un utilisateur a bien mentionné une suite de dix chiffres dans le champ « numéro de téléphone » d'un formulaire d'inscription

Le JavaScript supporte les expressions régulières via son objet **RegExp**,

- possède des propriétés et méthodes qu'on va pouvoir utiliser avec nos expressions régulières

L'intérêt des expressions régulières, est dans leur capacités de généricité et précision

- Correspondent à des séquences de caractères

# Syntaxe des regex

Pour construire une expression régulière, il suffit de

- préciser une séquence de caractères
- d'entourer cette séquence par des caractères d'encadrement
  - Vous pouvez choisir **n'importe quels caractères** d'encadrement,
  - écrivez le même caractère au début et à la fin de la regex.

Mais attention : Les expressions régulières utilisent beaucoup de caractères spéciaux pour permettre des recherches puissantes

- ces caractères signifient donc quelque chose de précis dans le langage des expressions régulières

Conseil: utiliser le slash (« / ») comme caractère d'encadrement,

- C'est un caractère qui ne va pas nous poser problème par la suite

# 1er exemple

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

 <body>
 <h1>Expressions régulières et JavaScript</h1>

 <script>
 var r = /Pierre/;
 </script>
 </body>
</html>
```

# Astuces

Mettre nos expressions régulières dans des variables, afin de pouvoir plus facilement les manipuler

Notre variable **r** contient notre première **regex** composée

- d'une séquence de caractères, en l'occurrence « Pierre »
- et de caractères d'encadrement (ou délimiteurs)

Pour le moment, cette expression régulière n'est pas utile

- Tout ce que l'on sait, c'est qu'on va travailler sur la séquence de caractères « Pierre »

Pour **rechercher** cette séquence, ou la **remplacer** par une autre par exemple, nous devrons utiliser les méthodes des objets **RegExp** et **String**

# Options

Notez qu'une expression régulière peut contenir des options

- vont être représentées par un caractère, comme le **i** par exemple
- Ont une signification spéciale dans le langage des regex et
- vont être les seuls caractères qui vont devoir être placés en dehors des délimiteurs, en fin de regex

Le caractère **i**, signifie que notre regex va ignorer la casse de notre séquence de caractères (« case-insensitive »)

Ainsi, on ne travaille plus seulement avec « Pierre », mais avec toutes ses déclinaisons de casse comme « pierre », « PIERRE », « PiErRe », etc.

## Le "i"

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

 <body>
 <h1>Expressions régulières et JavaScript</h1>

 <script>
 var r = /Pierre/i;
 </script>
 </body>
</html>
```

# RECHERCHES ET REMPLACEMENTS

Objets **RegExp** et **String** sous JavaScript

# Principales méthodes

Nous allons utiliser certaines méthodes des objets String et regExp afin justement d'effectuer de puissantes **recherches** ou des **remplacement**

Nous allons rechercher des types d'expression précis

- comme par exemple un enchaînement de 10 chiffres,
- la présence d'un « @ » et d'un « .com » dans une expression, etc.

Nous allons utiliser cinq méthodes :

- La méthode **match()** de l'objet String
- La méthode **search()** de l'objet String
- La méthode **replace()** de l'objet String
- La méthode **exec()** de l'objet regExp
- La méthode **test()** de l'objet regExp

# Méthode match() de l'objet String

Permet de rechercher des caractères dans une séquence de caractères

- prend en argument la séquence de caractères à rechercher

En précisant une regex en argument, nous allons pouvoir effectuer des recherches beaucoup plus poussées

- sans tenir compte de la casse par exemple

Cette méthode va renvoyer sous forme de tableau l'expression recherchée autant de fois qu'elle a été trouvée

Note: En l'utilisant avec une regex, elle ne renverra que la première occurrence trouvée

- sauf si nous ajoutons l'option g (pour effectuer une recherche globale) à notre regex

The screenshot shows a web browser window with the following details:

- Address bar: file:///Users/Pierre/Desktop/Cours%20Js
- Page title: Expressions régulières et JavaScript
- Text content:
  - Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !
  - Résultat match() sur regex 1 : Engagez
  - Résultat match() sur regex 2 : Engagez,engagez

```
<body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !</p>
 <p id="reg"></p>

 <script>
 /*On accède à notre paragraphe dans lequel on souhaite
 faire notre recherche/
 var expr = document.querySelector('p');

 //Première regex, avec l'option i
 var r1 = /engagez/i;

 //Deuxième regex, avec les options i et g
 var r2 = /engagez/ig;

 //On fait nos recherches avec match()
 var res1 = expr.textContent.match(r1);
 var res2 = expr.textContent.match(r2);

 //On affiche les résultats
 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Résultat match() sur regex 1 : ' + res1 +
 '
Résultat match() sur regex 2 : ' + res2;

 </script>
</body>
```

# Analyse de l'exemple précédent

nous mêlons ce que nous connaissons déjà sur le DOM avec nos regex pour bien comprendre l'ensemble

On commence par accéder à notre paragraphe puis on crée deux regex,

- une première avec l'option i
- et une seconde avec les deux options i et g

On utilise ensuite la méthode match() avec nos deux regex sur le contenu textuel de notre paragraphe

- Pour accéder à son contenu textuel, on utilise la propriété textContent
- on va rechercher à chaque fois la séquence « engagez » mais avec des options différentes

Si match() ne trouve aucun résultat, elle renverra la valeur **null**

# Méthode search() de l'objet String

La méthode **search()** retourne

- la position à laquelle a été trouvée la première occurrence de l'expression recherchée dans une chaîne de caractères
- **-1** si elle n'a pas été trouvée

On fournit une expression régulière en argument de cette méthode, (utilisation va très similaire à match() )

Exemple suivant:

- Une 1<sup>ère</sup> recherche de « engagez » sans tenir compte de la casse
  - search() s'arrête sur le premier Engagez se trouve en position 0
- Un 2<sup>ème</sup> recherche, sans l'option i donc en tenant compte de la casse.
  - ne va trouver la séquence que dans le mot « rengagez » en position 31



```
<body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !</p>
 <p id="reg"></p>

 <script>
 /*On accède à notre paragraphe dans lequel on souhaite
 faire notre recherche/
 var expr = document.querySelector('p');

 //Première regex, avec l'option i
 var r1 = /engagez/i;

 //Deuxième regex, sans option
 var r2 = /engagez/;

 //On fait nos recherches avec search()
 var res1 = expr.textContent.search(r1);
 var res2 = expr.textContent.search(r2);

 //On affiche les résultats
 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Résultat search() sur regex 1 : ' + res1 +
 '
Résultat search() sur regex 2 : ' + res2;
 </script>
</body>
```

# Méthode replace() de l'objet String

La méthode replace() va nous permettre de rechercher une expression et de la remplacer par une autre

- nous allons utiliser cette méthode avec nos regex pour remplacer plus efficacement

Prend 2 arguments :

- l'expression à rechercher (qui sera notre regex),
- et l'expression de remplacement

Exemple: on recherche la séquence "vous " dans notre chaîne de caractères et on la remplace par " moi "

- D'abord remarquer que seul le 1<sup>er</sup> "vous " a été remplacé car nous n'avons pas utilisé d'option avec notre regex
- en revanche, avec l'option g, tous les "vous" trouvés ont été remplacés

The screenshot shows a browser window with the following content:

- Title bar: file:///Users/Pierre/Desktop/Cours%20Js
- Page title: Expressions régulières et JavaScript
- Text content:
  - Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !
  - Résultat 1 : Engagez-moi qu'ils disaient, rengagez-vous qu'ils disaient !
  - Résultat 2 : Engagez-moi qu'ils disaient, rengagez-moi qu'ils disaient !

```
<body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !</p>
 <p id="reg"></p>

 <script>
 /*On accède à notre paragraphe dans lequel on souhaite
 faire notre recherche/
 var expr = document.querySelector('p');

 //Première regex
 var r1 = /vous/;

 //Deuxième regex, avec l'option g (global)
 var r2 = /vous/g;

 //On fait nos recherches avec replace()
 var res1 = expr.textContent.replace(r1, 'moi');
 var res2 = expr.textContent.replace(r2, 'moi');

 //On affiche les résultats
 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Résultat 1 : ' + res1 +
 '
Résultat 2 : ' + res2;
 </script>
</body>
```

# Méthode **test()** de l'objet regExp

La méthode **test()** va rechercher une séquence de caractères dans une chaîne de caractères

- Si la séquence est trouvée renvoie le booléen **true**
- Dans le cas contraire, elle renverra **false**

Notez qu'elle appartient à l'objet regExp et non plus à String

- Ainsi, on va l'utiliser sur un objet de type **RegExp**

Prend comme argument la chaîne de caractères dans laquelle faire la recherche

Exemple:

- Dans le 1<sup>er</sup> test, on recherche la séquence ENGAGEZ exactement, qui n'est pas trouvée par test().
  - La méthode renvoie donc false.
- Dans 2<sup>ème</sup> test, on cherche cette fois la séquence ENGAGEZ sans se soucier de la casse.
  - La méthode test() trouve donc un résultat, et renvoie true



```
<body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !</p>
 <p id="reg"></p>

 <script>
 //On accède au texte de notre paragraphe
 var rec = document.querySelector('p').textContent;

 //Première regex
 var r1 = /ENGAGEZ/;

 //Deuxième regex, avec l'option i (case insensitive)
 var r2 = /ENGAGEZ/i;

 //On fait nos recherches avec test()
 var res1 = r1.test(rec);
 var res2 = r2.test(rec);

 //On affiche les résultats
 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Résultat 1 : ' + res1 +
 '
Résultat 2 : ' + res2;
 </script>
</body>
```

# Méthode exec() de l'objet regExp

La méthode exec() va également rechercher une séquence de caractères dans une chaîne de caractères,

- mais va renvoyer le texte trouvé
- ou null si la recherche n'a pas été fructueuse

S'utilise avec un objet de type regExp et prend comme argument la séquence à rechercher

Notez que: si la séquence est trouvée plusieurs fois, celle-ci ne sera renvoyée qu'**une seule fois**

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

 <body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !</p>
 <p id="reg"></p>

 <script>
 //On accède au texte de notre paragraphe
 var rec = document.querySelector('p').textContent;

 var r1 = /aient/;

 //On fait notre recherche avec exec()
 var res1 = r1.exec(rec);

 //On affiche le résultat dans notre deuxième paragraphe
 var resultat = document.getElementById('reg');
 resultat.innerHTML = 'Résultat : ' + res1;
 </script>
 </body>
</html>
```

# Expressions régulières et JavaScript

Engagez-vous qu'ils disaient, rengagez-vous qu'ils disaient !

Résultat : aient

# Liste des quantificateurs

QUANTIFIEUR	SENS
a ?	On veut 0 ou 1 « a »
a+	On veut au moins un « a »
a*	On veut 0, 1 ou plusieurs « a »
^a	On veut un « a » en début de chaîne
a\$	On veut un « a » en fin de chaîne
a{X}	On veut une séquence de X « a »
a{X,Y}	On veut une séquence de X à Y fois « a »
a{X,}	On veut une séquence d'au moins X fois « a »
a(?=b)	On veut un « a » suivi d'un « b »
a(?!=b)	On veut un « a » non suivi d'un « b »

```

<body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>On apprend à utiliser les regex en JavaScript</p>
 <p id="reg"></p>

 <script>
 //On accède au texte de notre paragraphe
 var rec = document.querySelector('p').textContent;

 //On veut savoir si notre paragraphe contient un "x"
 var r1 = /x+/";
 var res1 = r1.test(rec);

 /*On cherche "e" ou "end" (on utilise les parenthèses)
 pour faire porter "?" sur "nd"/
 var r2 = /e(nd)?/g;
 var res2 = rec.match(r2);

 //Notre paragraphe finit par "e" ?
 var r3 = /e$/i;
 var res3 = r3.test(rec);

 //Notre paragraphe contient-il "pp"
 var r4 = /p{2}/;
 var res4 = r4.test(rec);

 //Y a t-il l'expression "Java" suivie de "script" ?
 var r5 = /Java(?:Script)/;
 var res5 = r5.test(rec);

 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Présence d\'un "x" : ' + res1 +
 '
Occurrences de "e" ou "end" : ' + res2 +
 '
finit par "e" ? ' + res3 +
 '
Contient deux "p" à la suite ? : ' + res4 +
 '
Présence de "Java" suivi de "Script" ? : ' + res5;
 </script>
</body>

```

file:///Users/Pie

# Expressions régulières

On apprend à utiliser les regex en JavaScript

Présence d'un "x" : true  
 Occurrences de "e" ou "end" : end,e,e,e,e  
 finit par "e" ? false  
 Contient deux "p" à la suite ? : true  
 Présence de "Java" suivi de "Script" ? :true

# Options ou modificateurs

Il existe 3 options : l'option **i**, l'option **g** et l'option **m**.

L'option **i** nous permet de ne pas tenir compte de la casse dans notre regex,

L'option **g** sert à exécuter des recherches globales : plutôt que de s'arrêter au premier résultat trouvé,

L'option **m**, va nous permettre d'effectuer notre recherche sur plusieurs lignes.

- **m** va considérer chaque retour à la ligne comme la fin d'une première ligne et le début d'une deuxième ligne.
- Nous allons donc généralement utiliser l'option m avec les quantificateurs ^ et \$ qui vont alors considérer chaque retour à la ligne comme la fin d'une première chaîne et le début d'une autre.

```
<h1>Expressions régulières et JavaScript</h1>
<p>On apprend à utiliser les regex en JavaScript</p>
<p id="reg"></p>
```

```
<script>
 //On accède au texte de notre paragraphe
 var rec = document.querySelector('p').textContent;
 var rec2 = 'Je suis sur \ndeux lignes en Js'
```

```
//On cherche "on", "On", "oN" ou "ON"
var r1 = /on/i;
var res1 = rec.match(r1);
```

```
//On cherche tous les 'e'
var r2 = /e/g;
var res2 = rec.match(r2);
```

```
//On cherche si une ligne commence par "d" (sans m) dans rec2
var r3 = /^d/;
var res3 = r3.test(rec2);
```

```
//On cherche si une ligne commence par "d" (avec m) dans rec2
var r4 = /^d/m;
var res4 = r4.test(rec2);
```

```
var resultat = document.getElementById('reg');
resultat.innerHTML =
 'Résultat regex 1 : ' + res1 +
 '
Résultat regex 2 : ' + res2 +
 '
Résultat regex 3 : ' + res3 +
 '
Résultat regex 4 : ' + res4;
```

&lt;/script&gt;

&lt;/body&gt;



The screenshot shows a browser window with the title bar "file:///Users/Pie". The main content area displays the following text:

# Expressions régulières

On apprend à utiliser les regex en JavaScript

Résultat regex 1 : On

Résultat regex 2 : e,e,e,e,e,e

Résultat regex 3 : false

Résultat regex 4 : true

# Classes de caractères

vont nous permettre de mentionner des plages de caractères pour effectuer nos recherches

On utilise une paire de crochets ouvrant et fermant, [ ]

- sauf dans un cas particulier, nous utiliserons des parenthèses ()



```

<body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>On apprend à utiliser les regex en JavaScript</p>
 <p id="reg"></p>

 <script>
 //On accède au texte de notre paragraphe
 var rec = document.querySelector('p').textContent;

 //On cherche "aeiouy"
 var r1 = /aeiouy/g;
 var res1 = rec.match(r1);

 //On cherche tous les 'a', 'e', 'i', 'o', 'u' et 'y'
 var r2 = /[aeiouy]/g;
 var res2 = rec.match(r2);

 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Recherche sans classe : ' + res1 +
 '
Recherche avec classe : ' + res2;
 </script>
</body>
</html>

```

# Classes de caractères: Exemples

CLASSE	SENS
[abcde]	Trouve tous les caractères à l'intérieur des crochets
[^abcde]	Trouve tout caractère ne se situant pas entre les crochets
[a-z]	Trouve n'importe quelle lettre entre a et z
[^a-z]	Trouve n'importe quel caractère qui n'est pas une lettre minuscule de l'alphabet
[0-9]	Trouve n'importe quel nombre entre 0 et 9
[^0-9]	Trouve n'importe quel caractère qui n'est pas un nombre compris entre 0 et 9
(jour soir)	Trouve jour et soir

- Attention à l'utilisation du symbole ^ à l'intérieur des classes de caractères
  - ne possède pas la même signification que dans nos regex classiques

```

<body>
 <h1>Expressions régulières et JavaScript</h1>
 <p>On apprend à utiliser les regex en JavaScript en 2015</p>
 <p id="reg"></p>

 <script>
 //On accède au texte de notre paragraphe
 var rec = document.querySelector('p').textContent;

 //On cherche toutes les majuscules entre C et X
 var r1 = /[C-X]/g;
 var res1 = rec.match(r1);

 //On cherche n'importe quel caractère sauf a, e, i, o, u
 var r2 = /^[^aeiou]/g;
 var res2 = rec.match(r2);

 //On cherche tous les chiffres entre 0 et 9
 var r3 = /[0-9]/g;
 var res3 = rec.match(r3);

 /*On cherche si on a bien une majuscule en début de chaîne.
 Attention : le "^" est bien en dehors de notre classe/
 var r4 = /^[A-Z]/;
 var res4 = r4.test(rec);

 //On cherche toute succession de quatre chiffres
 var r5 = /[0-9]{4}/;
 var res5 = rec.match(r5);

 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Majuscules entre C et X : ' + res1 +
 '
Tout sauf voyelles minuscules : ' + res2 +
 '
Tous les chiffres : ' + res3 +
 '
Majuscule en début de chaîne ? : ' + res4 +
 '
Succession de 4 chiffres : ' + res5;
 </script>
</body>
</html>

```



# Recherche dans plusieurs intervalles

```
<script>
 //On accède au texte de notre paragraphe
 var rec = document.querySelector('p').textContent;

 //On cherche tout entre a et z plus "à", "é" et "è"
 var r1 = /[a-zAÉÈ]/g;
 var res1 = rec.match(r1);

 //On cherche tout entre a et z, A et Z et 0 et 9
 var r2 = /[a-zA-Z0-9]/g;
 var res2 = rec.match(r2);

 /*On cherche tout ce qui n'est pas dans a-z ni A-Z
 Les espaces vont donc être comptés/
 var r3 = /[^\w]/g;
 var res3 = rec.match(r3);

 var resultat = document.getElementById('reg');
 resultat.innerHTML =
 'Lettre min plus "à", "é" et "è" : ' + res1 +
 '
Lettre min, maj et chiffre : ' + res2 +
 '
Tout sauf a-z et A-Z : ' + res3;
</script>
</body>
</html>
```

# Méta-caractères, classes abrégées

META CARACTERE	SENS
.	Seul méta-caractère ne possédant pas d'antislash. Trouve tout caractère, sauf un retour à la ligne
\w	Trouve toute lettre. Équivalent à [a-zA-Z]
\W	Trouve tout ce qui n'est pas une lettre. Équivalent à [^a-zA-Z]
\d	Trouve n'importe quel chiffre. Équivalent à [0-9]
\D	Trouve tout sauf un chiffre. Équivalent à [^0-9]
\s	Trouve un espace
\S	Trouve tout sauf un espace
\ba	Trouve tout « a » situé en début ou en fin d'un mot
\Ba	Trouve tout « a » non situé en début ou en fin de mot
\n	Trouve un retour à la ligne (le « \n » en JavaScript)

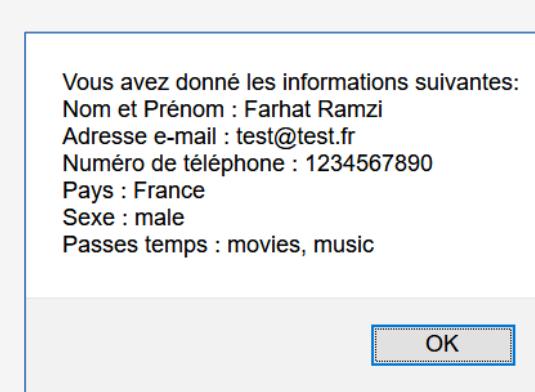
# Exercice

- **Objectif :**

- Création d'un formulaire dynamique dont une partie des vérifications est effectuée par le Javascript, côté client.

- **Conditions à respecter :**

- Nom et Prénom : au moins deux mots et au plus trois mots
- E-mail : des caractères (pas d'espaces), puis un @ puis des caractères (pas d'espaces), puis un point, puis des caractères (pas d'espaces)
- Numéro de téléphone : 10 chiffres dont le premier à gauche non 0
- Pays: choix obligatoire d'un pays
- Sexe: choix obligatoire de sexe
- Passes temps : optionnels
- **Exemple de résultat:**



## Formulaire d'inscription

Nom et Prénom

Adresse e-mail

Numéro de téléphone

Pays

Sexe

Homme  Femme

Centres d'intérêt (Optionnel)

Sport  Cinéma  
 Musique

**Inscription**

# Programme de formation

I- Les technologies du Web

II- Le langage JavaScript, prise en mains

III- Evénements et données

IV- Gestion de formulaires HTML

V- Interaction avec les CSS

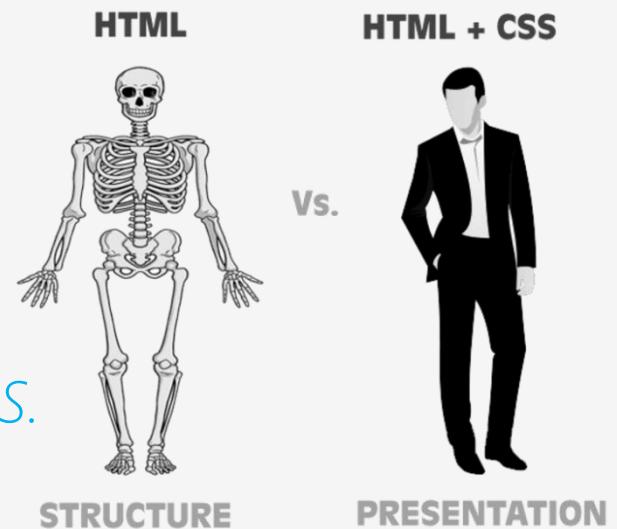
VI- Manipulation du DOM XML

VII- Ajax

## V- Interaction avec les CSS

1. Rappel sur les feuilles de style en cascade (CSS-1, CSS-2). Les outils pour les manipuler.
2. Implémentation des CSS en tant que propriétés des objets du DOM.
3. Modification directe des propriétés CSS des objets du DOM.
4. Modification de l'objet CSS stylesheets.
5. Rendre la page dynamique via le changement des propriétés de style.

**Exercice:** *Familiarisation avec les feuilles de style et à leur manipulation JS.*



# CSS Introduction

Il existe trois façons d'insérer une feuille de style : "myStyle.css"

## 1. CSS externe

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

h1 {
 color: navy;
 margin-left: 20px;
}
```

## 2. CSS interne

```
<html>
 <head>
 <style>
 body {
 background-color: linen;
 }
 h1 {
 color: maroon;
 margin-left: 40px;
 }
 </style>
 </head>
```

## 3. CSS dans les balises

```
<!DOCTYPE html>
<html>
 <head>
 <link rel="stylesheet"
 href="myStyle.css">
 </head>
 <body>

 <h1>This is a heading</h1>
 <p>This is a paragraph.
 </p>

 </body>
</html>
```

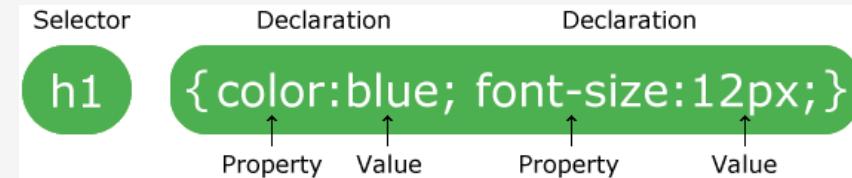
This is a heading

This is a paragraph.

```
<h1 style="color:blue;text-align:center;">
</h1>
<p style="color:red;">T.
</p>
```

# Sélecteurs CSS

Syntaxe:


[DEMO](#)

Sélecteur	Exemple	description
#id	#firstname	Sélectionne l'élément avec id="firstname"
.class	.intro	Sélectionne tous les éléments avec class="intro"
element.class	p.intro	Sélectionne uniquement <p> éléments avec class="intro"
*	*	Sélectionne tous les éléments
element	p	Sélectionne tous les <p>éléments
element, element,..	div, p	Sélectionne tous les <div> et tous les <p>éléments
element element	p a	appliquer des styles à tous les éléments a contenus dans des éléments p
element> element	Body> a	tous les liens (éléments a) qui sont des enfants directs de l'élément body
Element + element	E + F	Sélectionne tout élément F placé directement après un élément E
Element ~ element	E~F	Sélectionne tout élément F placé après un élément E dans la page

# Exemple

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 <link rel="stylesheet" href="style.css">
 </head>
 <body>
 <h1 class="gros souligne orange">Un titre de niveau 1</h1>
 <p class="bleu">Un premier paragraphe</p>
 <p>Un autre paragraphe avec un lien</p>

 Elément de liste 1
 <li class="souligne">Elément de liste 2

 <h2 class="gros italique bleu">Un titre de niveau 2</h2>
 </body>
</html>
```

```
.gros{
 font-size: 24px;
}
/*Texte en italique*/
.italique{
 font-style: italic;
}
/*Crée un trait de soulignement sous le texte*/
.souligne{
 text-decoration: underline;
}
.orange{
 color: orange;
}
.bleu{
 color: blue;
}
.vert{
 color: green;
}
```

Un titre de niveau 1

Un premier paragraphe

Un autre paragraphe avec un [lien](#)

- Elément de liste 1
- Elément de liste 2

*Un titre de niveau 2*

# CSS: exemple 2



```
<!DOCTYPE html>
<html>
 <head>
 <title>Sélecteurs CSS</title>
 <meta charset= "utf-8">
 <link rel="stylesheet" href="styles.css">
 </head>

 <body>
 <!--On donne la même class à notre titre et à un paragraphe-->
 <h1 class="p1">Un titre de niveau 1</h1>
 <p id="p1">Un paragraphe contenant</p>
 <p class="p1">Un deuxième paragraphe</p>
 <p>Un troisième paragraphe</p>
 </body>
</html>

/*L'élément portant l'id "p1" sera en bleu*/
#p1{
 color: blue;
}

/*Les éléments portant la class "p1" seront en rouge*/
.p1{
 color: red;
}
```

# Exemples de propriétés CSS

Exemples de systèmes de couleurs:

```
<h1 style="background-color:rgb(255, 99, 71);"> ... </h1>
<h1 style="background-color:#ff6347;"> ... </h1>
<h1 style="background-color:hsl(9, 100%, 64%);"> ... </h1>
```

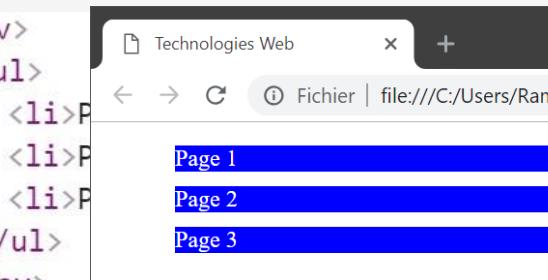
Arrière-plan ou background

```
background: rgba(0, 128, 0, 0.3) /* Green background with 30% opacity */
background-color: green;
opacity: 0.3;
background-image: url("paper.gif");
```

Affichage ou **display**

- **block** : retour à la ligne au début de l'élément et à la fin de l'élément
- **inline** : continuer sur la même ligne
- **inline-block** : c'est comme inline mais avec la possibilité de fixer l'hauteur et la largeur de l'élément
- **none** : cacher l'élément

etc.

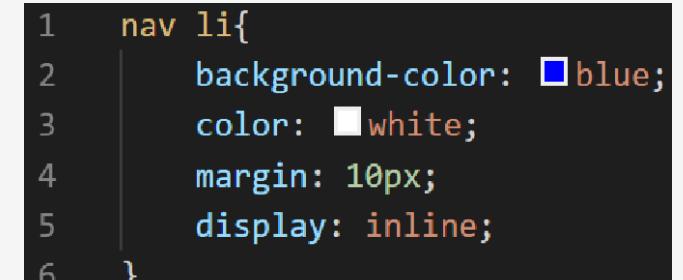


```
<nav>

 Page 1
 Page 2
 Page 3

</nav>
```

```
1 nav li{
2 background-color: blue;
3 color: white;
4 margin: 10px;
5 }
```



```
1 nav li{
2 background-color: blue;
3 color: white;
4 margin: 10px;
5 }
```

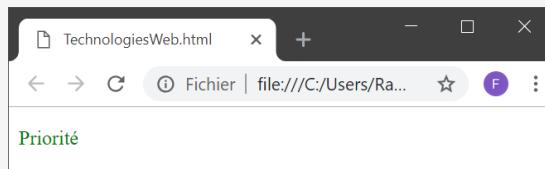
```
1 nav li{
2 background-color: blue;
3 color: white;
4 margin: 10px;
5 }
```

# Priorité des règles

## □ Ordre dans la même déclaration

1. *Id*
2. *Classe*
3. *Balise*
4. *Dernier si égalité*
5. *Possible indication de priorité*

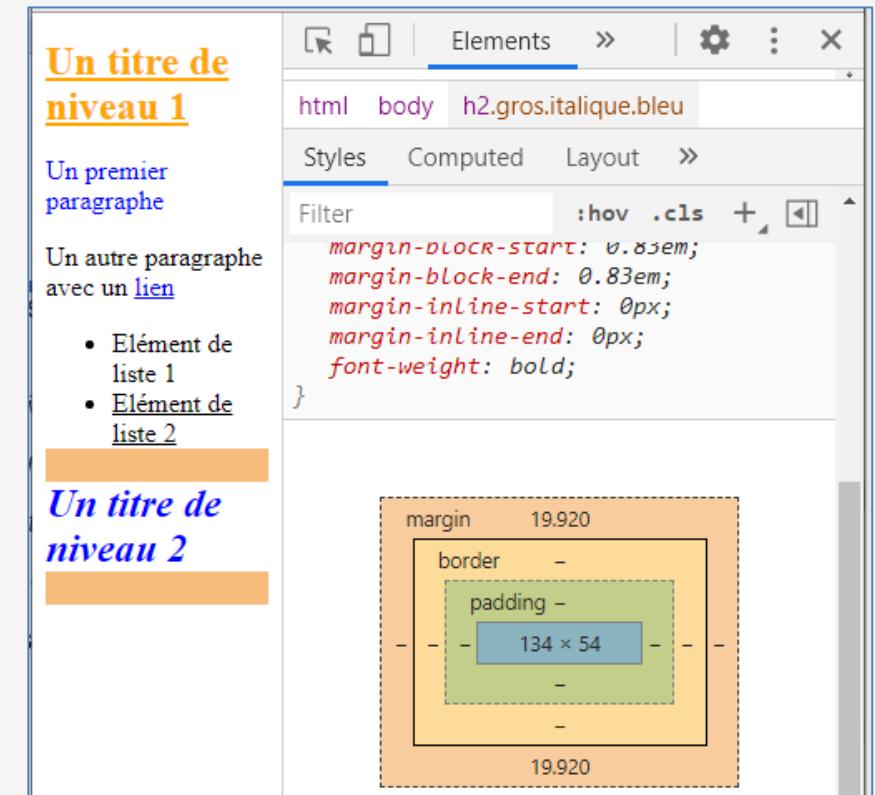
```
5 TechnologiesWeb.html x
1 <!DOCTYPE html>
2 <html lang="fr">
3 <head>
4 <meta charset="UTF-8">
5 <link rel="stylesheet" href="TechnologiesWeb.css">
6 </head>
7 <body>
8 <p id="monID" classe="maClasse">Priorité</p>
9 </body>
10 </html>
```



```
3 TechnologiesWeb.css x
1 .maClasse{
2 color: blue
3 }
4 #monID{
5 color: green;
6 }
7 p{
8 color: blueviolet;
9 }
```

# Modèle de boîte

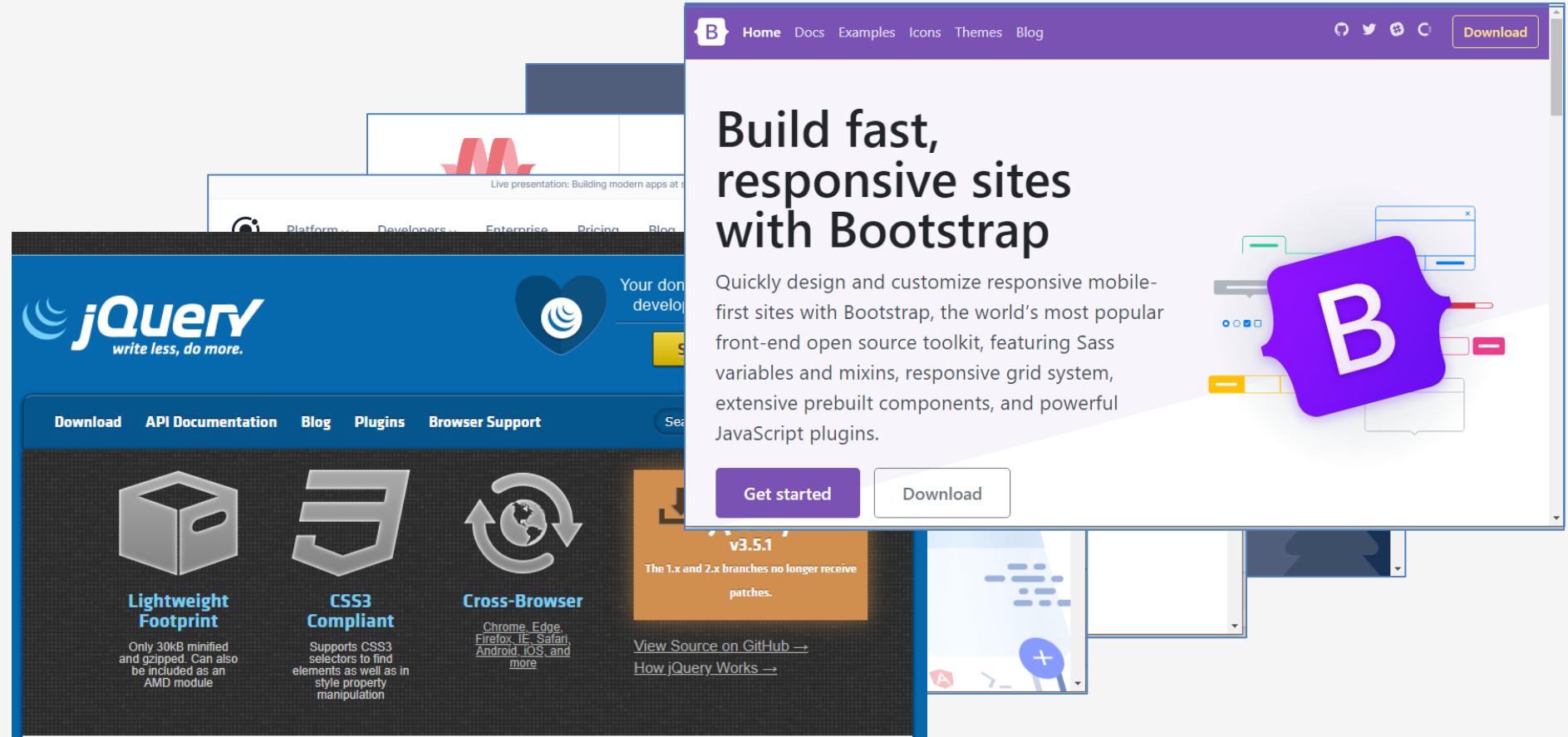
- Tout élément HTML est considéré comme une boîte
- Chaque boîte à :
  - Un contenu (texte, image, vidéo, etc.) : **content**
  - Une bordure (contour autour de l'élément) : **border**
  - Un rembourrage (espace entre le contenu et la bordure) : **padding**
  - Une marge (espace entre la bordure et les éléments adjacents) : **margin**
- → Possibilité de contrôler ces propriétés via le CSS



# CSS Frameworks

Une combinaison très riche de **JS**, **CSS**, d'images vectrielles, etc.

- Bootstrap
- Font-Awesome
- Ionic
- Materialized
- jQuery
- etc.



# méthodes querySelector() et querySelectorAll()

Ces méthodes vont nous permettre d'accéder à des éléments HTML correspondant à certain sélecteur CSS:

- un id,
- une class,
- un type d'élément,
- un attribut

**querySelector()** va renvoyer des informations relatives au premier élément trouvé correspondant au sélecteur CSS sélectionné,

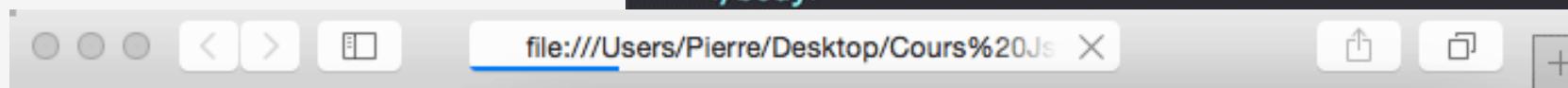
**querySelectorAll()** va renvoyer des informations sur tous les éléments correspondants.

```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On accède au premier élément de type lien contenu dans un paragraphe
 var lien = document.querySelector('p a');

 //On accède à tous les paragraphes portant la class "para"
 var p = document.querySelectorAll('.para');

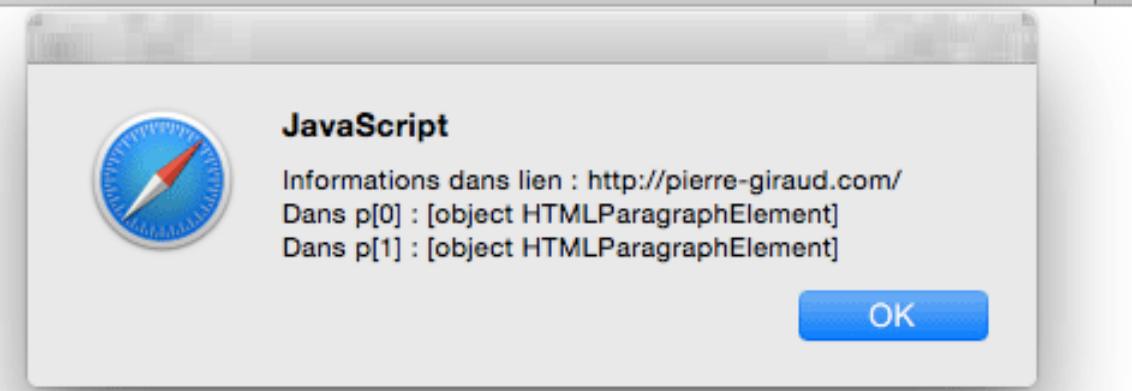
 //On affiche les résultats obtenus
 alert('Informations dans lien : ' + lien +
 '\nDans p[0] : ' + p[0] +
 '\nDans p[1] : ' + p[1])
 </script>
</body>
```



# Le DOM

Du texte et un lien

Un deuxième paragraphe



# Modifier le CSS d'un élément

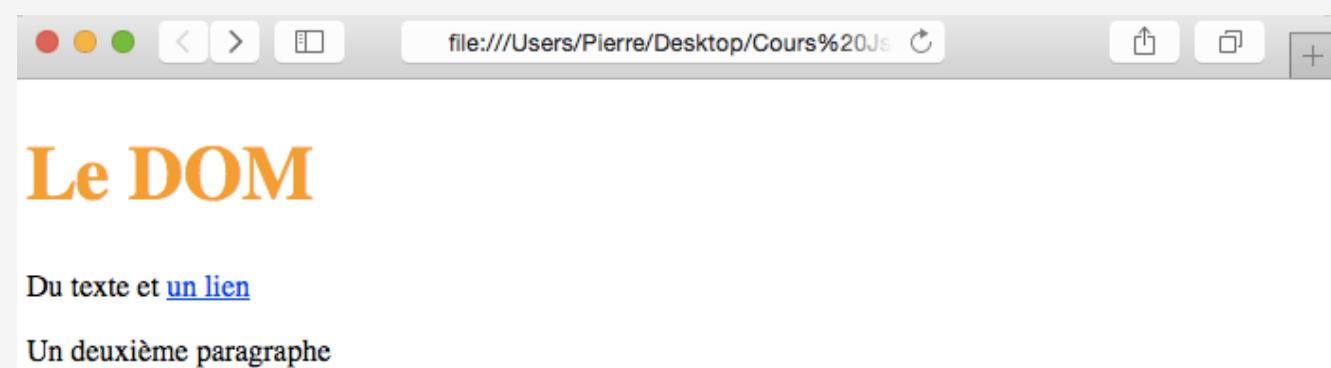
Dans le code précédent:

- on utilise **querySelector()** pour accéder au premier lien rencontré contenu dans un paragraphe (en utilisant le sélecteur CSS **p a**)
- on utilise **querySelectorAll()** pour accéder à tous les paragraphes disposant d'un attribut class= "para"(sélecteur CSS **.para**).

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 // On veut que notre titre s'affiche en orange
 document.getElementById('gros_titre').style.color = 'orange';

 //On attribue une taille de 40px à notre titre
 document.getElementById('gros_titre').style.fontSize = '40px';
 </script>
 </body>
</html>
```



# Exercice

Vous disposez d'un formulaire qui permet de recevoir:

- un nom de personne sous forme de chaîne de caractère
- Et un chiffre qui sera relative à une somme d'argent à recevoir ou à dépenser (signe - accepté)
- Un bouton de validation des saisies

Selon les saisies: Si elle sont pas vides et qu'elles ont les bon formats, des colorations et icônes **fontawesome** adéquates pourront apparaître:

- (✓) Vert → bonne saisie
- (X) Rouge → mauvaise saisie

The form consists of two input fields and a button. The first input field is labeled 'Nom' and contains the value 'Bassem', which is followed by a green checkmark icon. The second input field is labeled 'Montant (- dépenses, + revenus)' and contains the value '40', which is preceded by a small up-down arrow icon and followed by a green checkmark icon. Below the inputs is a large blue button with the text 'Ajouter'.

# Programme de formation

I- Les technologies du Web

II- Le langage JavaScript, prise en mains

III- Evénements et données

IV- Gestion de formulaires HTML

V- Interaction avec les CSS

VI- Manipulation du DOM XML

VII- Ajax

# VI- Manipulation du DOM , DOM XML

# JS



1. Interaction JS avec le DOM
2. Les objets du DOM (window, document...) et leur manipulation
3. Manipulation des objets du DOM (lecture, ajout, suppression, modification de noeuds).
4. Implémentation des CSS en tant que propriétés des objets du DOM.
5. Modification directe des propriétés CSS des objets du DOM.
6. Présentation du langage XML (éléments attributs). Implémentation des parseurs XML chez Microsoft IE et les autres : variantes entre les navigateurs, maintenance...
7. Manipulation des objets du DOM (lecture, ajout, suppression, modification de noeuds).

**Exercice:** Familiarisation à la construction d'un chemin d'accès à un élément du DOM.

# Utilité du DOM

Document Object Model, ou **DOM**

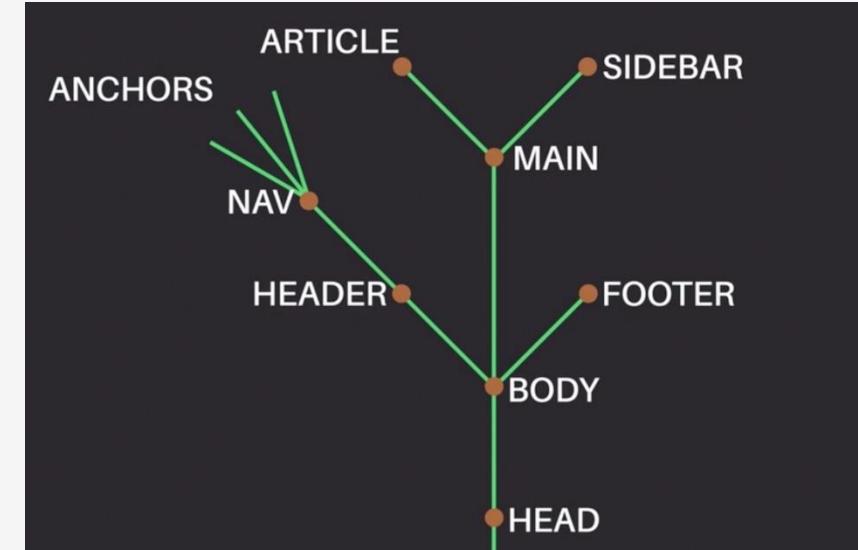
- une partie pour le XML,
- une partie pour le **HTML**
- et une partie cœur

Créé automatiquement par votre navigateur lors du chargement de la page.

C'est un standard de programmation reconnu par tous les navigateurs:

- considère les éléments HTML comme des objets qui possèdent:
  - des propriétés
  - et des méthodes
- nous permet de déclencher des événements

Il nous permettra de manipuler du code HTML (et CSS) avec le JavaScript



# La structure du DOM HTML

Tout ce qui est présent dans une page HTML va être considéré comme un nœud, ou « **node** »:

- le document HTML lui même,
- les éléments HTML,
- les attributs HTML,
- le texte à l'intérieur des éléments,
- etc.

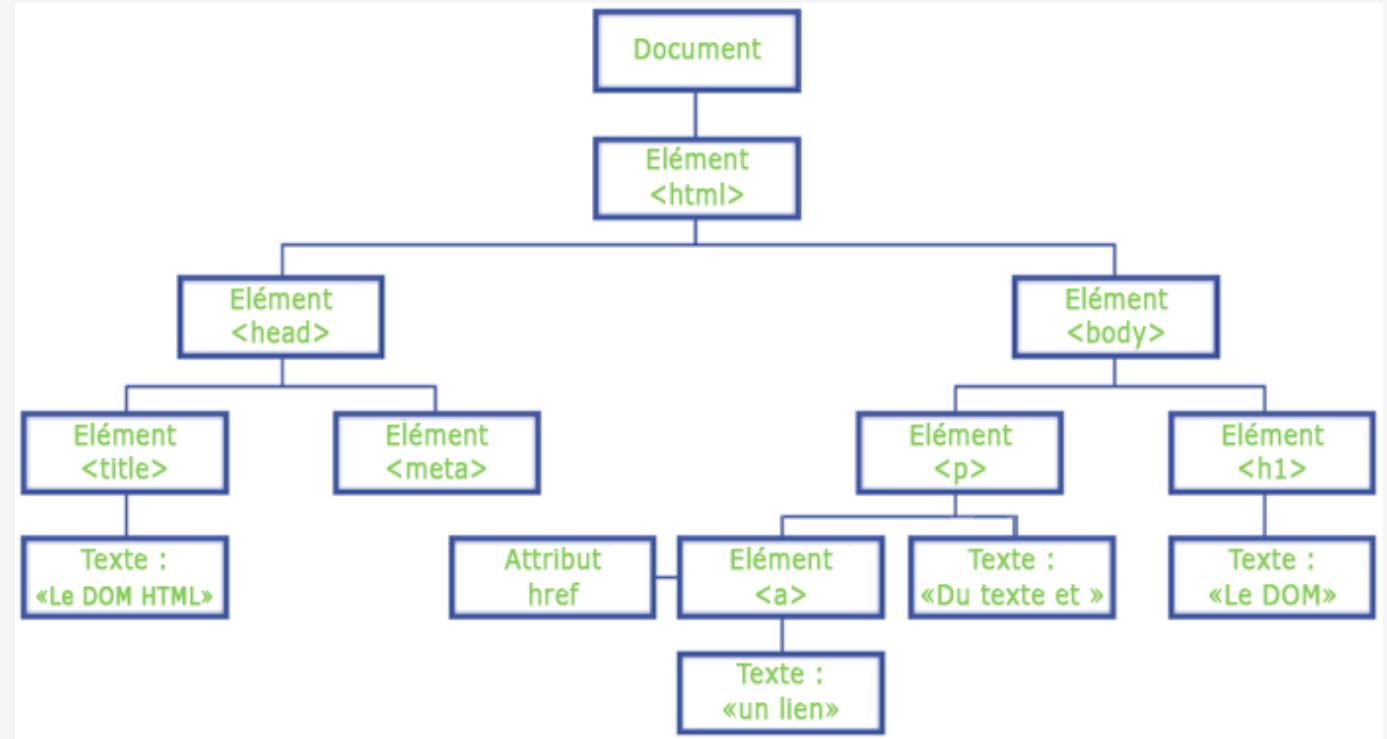
Plusieurs types de nœuds existent pour les objets HTML :

- le type **ELEMENT\_NODE** (pour les éléments HTML),
- le type **TEXT\_NODE** (pour le texte),
- etc.

Construit comme une hiérarchie de nœuds (**Arbre**)

# DOM définition

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien </p>
 </body>
</html>
```



- Dans le schéma précédent, un premier nœud de type **élément** est représenté par l'élément html.
- Ce nœud possède deux enfants : **head** et **body** qui sont de type **Element** et qui possèdent eux mêmes d'autres enfants.
- head et body sont des nœuds « **frères** » : ils ont le même parent mais il n'y a pas de relation de hiérarchie entre eux.
- le texte contenu dans les éléments est ici reconnu comme un nœud à part entière
  - On peut donc le manipuler avec des méthodes et propriétés en JavaScript.

# L'objet document et ses méthodes

Une des forces de JavaScript est de permettre de manipuler des éléments HTML en se servant du DOM.

Nous allons toujours devoir passer par l'objet Document

Nous allons voir différentes méthodes de l'objet **Document**:

- La méthode getElementById() ;
- La méthode getElementsByTagName() ;
- La méthode getElementsByClassName() ;
- La méthode querySelector() ;
- La méthode querySelectorAll().

# La méthode getElementById()

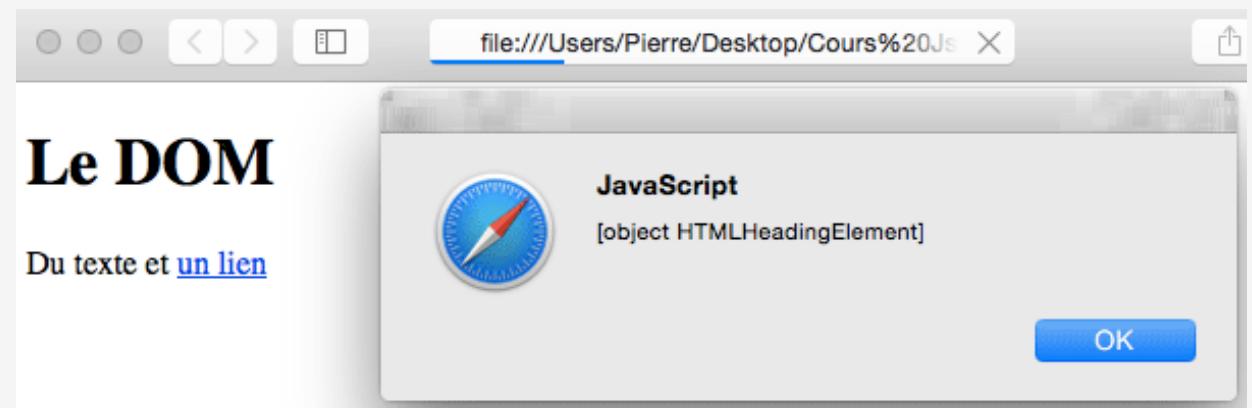
- Permet de cibler un élément HTML possédant un attribut **id** en particulier si l'élément est trouvé, getElementById() va renvoyer l'élément en tant qu'objet.
  - Sinon, elle renverra la valeur **null**.

Exemple :

- on accède à l'élément HTML de notre page possédant l'id= "**gros\_titre**"
- on stocke les informations renvoyées dans une variable **titre**.

```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p>Du texte et un lien</p>

 <script>
 /*On utilise la méthode getElementById de l'objet document.
 On enferme le résultat renvoyé dans une variable "titre"/
 var titre = document.getElementById('gros_titre');
 alert(titre);
 </script>
</body>
```



# getElementsByName()

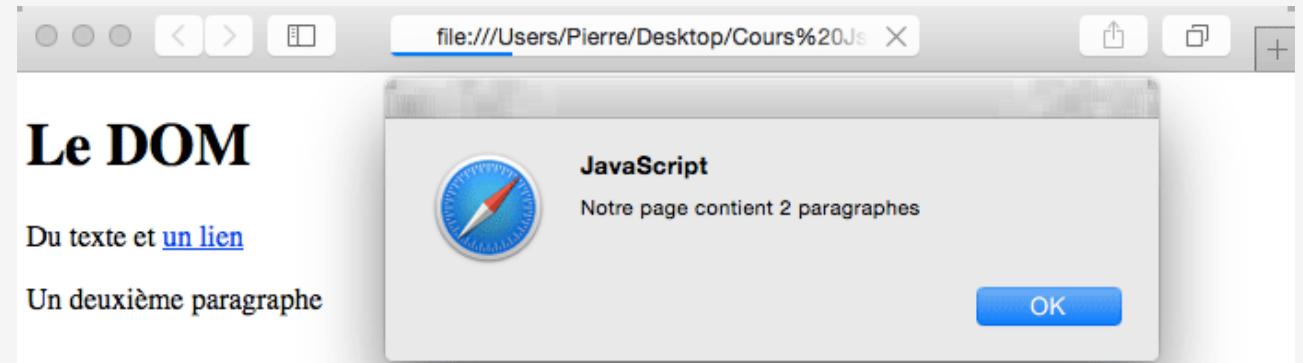
Elle va retourner des informations relatives à tous les éléments HTML d'un même « genre »

- Les éléments sont placés dans un tableau.
- tous les éléments **<p>** par exemple

Son intérêt est qu'on va ensuite pouvoir récupérer un élément en question en utilisant un indice du tableau créé.

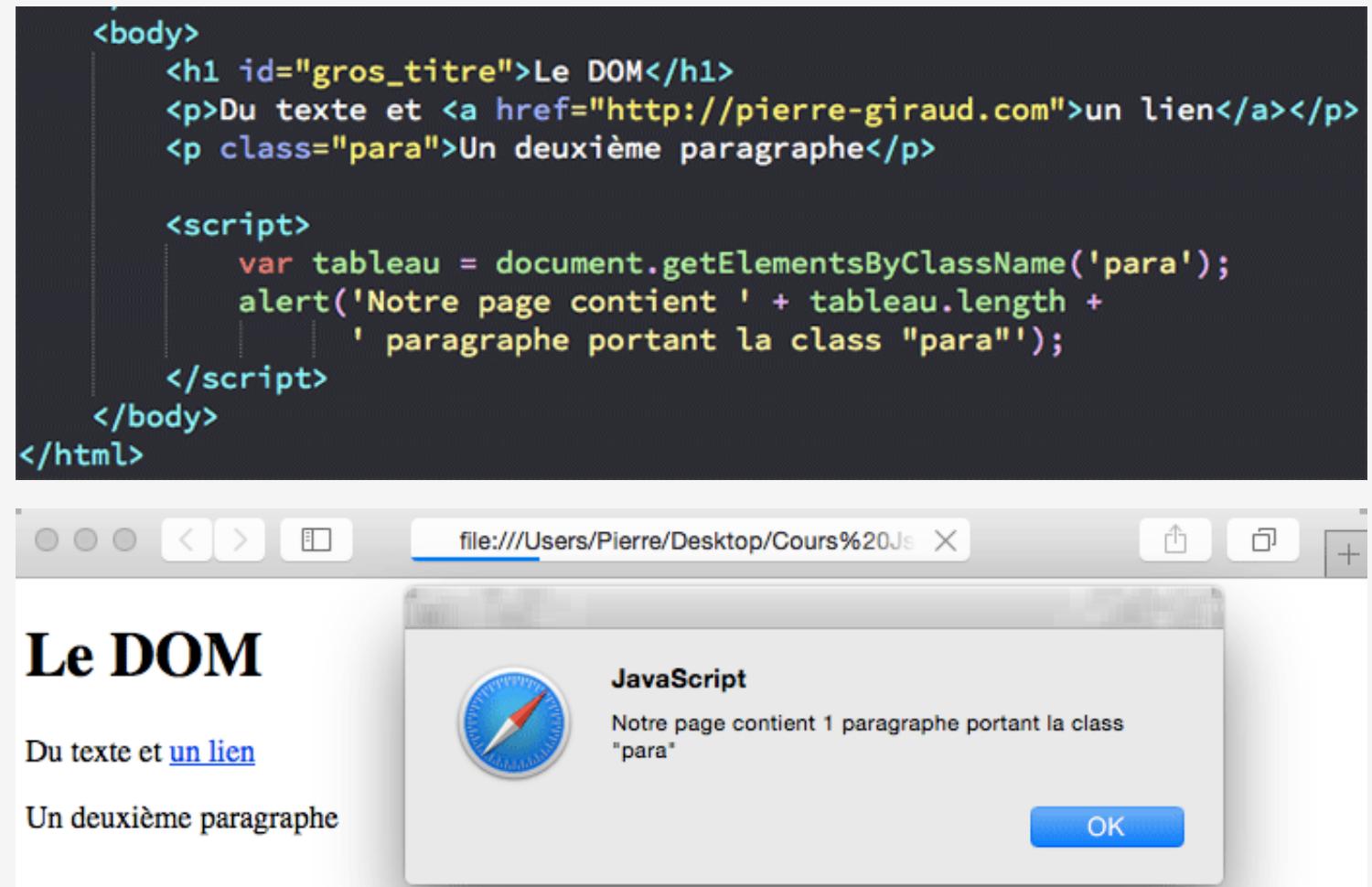
```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p>Du texte et un lien</p>
 <p>Un deuxième paragraphe</p>

 <script>
 var tableau = document.getElementsByTagName('p');
 alert('Notre page contient ' + tableau.length + ' paragraphes');
 </script>
</body>
</html>
```



# getElementsByClassName()

- Elle va nous permettre d'accéder aux éléments HTML disposant d'un attribut class en particulier.
- **getElementsByClassName()** va prendre la valeur d'un attribut class en argument.
- Elle permet de renvoyer également un tableau.



The screenshot shows a browser window with the following code in the source view:

```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p>Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 var tableau = document.getElementsByClassName('para');
 alert('Notre page contient ' + tableau.length +
 ' paragraphe portant la class "para"');
 </script>
</body>
</html>
```

The browser title bar reads "file:///Users/Pierre/Desktop/Cours%20Js". The main content area displays the heading "Le DOM" and two paragraphs: "Du texte et [un lien](http://pierre-giraud.com)" and "Un deuxième paragraphe". A JavaScript alert box is overlaid on the page, containing the message "Notre page contient 1 paragraphe portant la class 'para'".

# Accéder au contenu des éléments HTML et au texte

- Jusqu'à présent, nous avons pu accéder à des éléments HTML et retourner des informations liées aux objets
- On va également pouvoir accéder au contenu des éléments HTML:
  - ce qui se situe entre les balises ouvrante et fermante d'un élément.
  - La manière la plus simple de procéder est d'utiliser:
  - La propriété **innerHTML** sur le résultat renvoyé par nos méthodes.

```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 /*On accède à notre élément h1 avec la méthode getElementById() puis
 *on récupère son contenu grâce à la propriété innerHTML.
 On stocke le tout dans une variable puis on affiche son contenu/
 var titre = document.getElementById('gros_titre').innerHTML;
 alert(titre);
 </script>
</body>
</html>
```

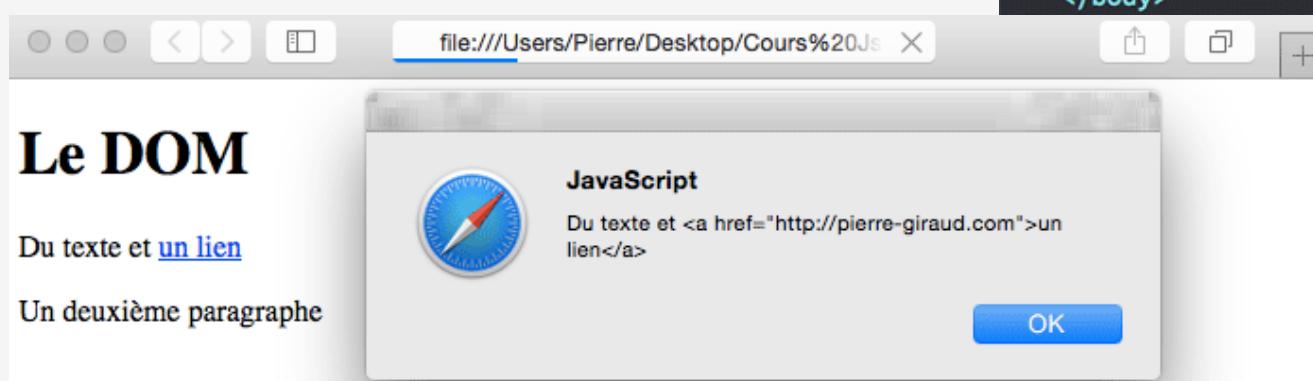


Attention : **innerHTML** va récupérer tout le contenu d'un élément HTML et pas seulement le texte contenu

Par exemple, si on applique **innerHTML** en ciblant notre paragraphe contenant un lien, la propriété va nous retourner tout le code du lien

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 /*On accède au premier paragraphe portant la class "para" et
 on récupère ce qui se trouve à l'intérieur*/
 var p = document.querySelector('.para').innerHTML;
 alert(p);
 </script>
 </body>
```



- Si on souhaite ne récupérer que le contenu textuel présent dans un élément, on utilise la propriété **textContent**.
- Notez qu'on applique les propriétés innerHTML et textContent sur des nœuds de type Element et non pas sur le document en soi
- Car innerHTML et textContent sont des propriétés de l'objet **Element** et non pas **Document**.

```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 /*La différence entre innerHTML et textContent*/
 var p = document.querySelector('.para').innerHTML;
 var t = document.querySelector('.para').textContent;

 alert('Contenu récupéré avec innerHTML : \n' + p +
 '\n\nContenu récupéré avec textContent : \n' + t);
 </script>
</body>
</html>
```



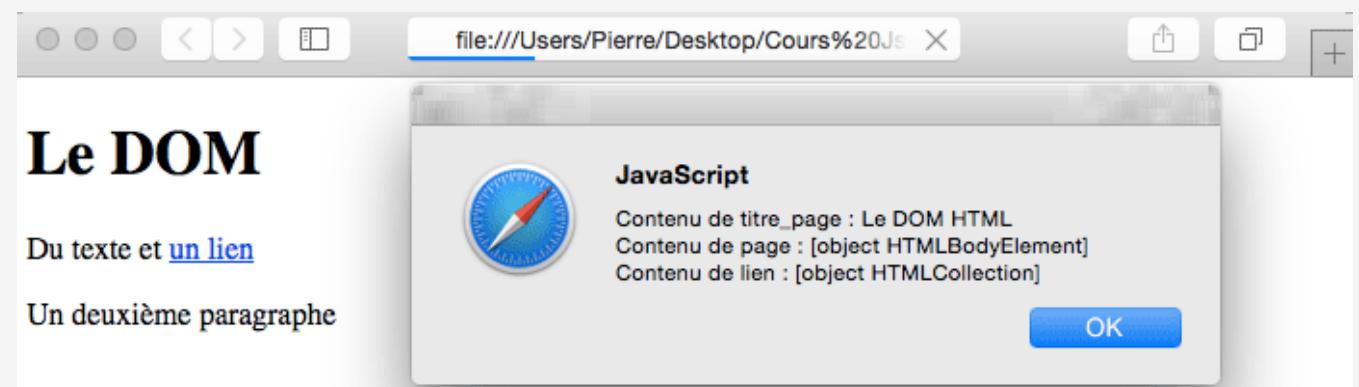
# Accéder directement à des types d'éléments

- Le DOM nous offre la possibilité, via certaines de ses propriétés, d'accéder directement à un type ou une collection d'éléments HTML en particulier.
- Par exemple, on va pouvoir accéder à
- l'élément body grâce à la propriété du même nom
- un élément de type lien grâce à la propriété links.
- Nous allons voir dans le code suivant les propriétés
  - title,
  - body et
  - links

```
<body>
<h1 id="gros_titre">Le DOM</h1>
<p class="para">Du texte et un lien</p>
<p class="para">Un deuxième paragraphe</p>

<script>
 var titre_page = document.title;
 var page = document.body;
 var lien = document.links;

 alert('Contenu de titre_page : ' + titre_page +
 '\nContenu de page : ' + page +
 '\nContenu de lien : ' + lien);
</script>
</body>
</html>
```



# Modification des éléments HTML

Usage de JS

# Les objets JS document et element

la propriété **innerHTML** appartient bien à l'objet **Element**, et non pas à Document.

L'objet **Element** hérite de son parent Document et rajoute ses méthodes et propriétés

On écrira **document.getElementById()** pour accéder à un élément HTML dans une page.

On appliquera les propriétés et méthodes de l'objet **Element** sur les éléments HTML

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 /*On peut procéder en deux temps, en récupérant d'abord les informations
 *liées à un élément en particulier dans une variable, puis appliquer
 innerHTML à cette variable par la suite/
 var titre = document.getElementById('gros_titre');
 var texteTitre = titre.innerHTML;

 /*Ou on peut accéder au texte d'un élément en un coup, sans utiliser de
 variable/
 var textePara = document.querySelector('.para').textContent;

 alert('Titre : ' + texteTitre +
 '\nParagraphe 1 : ' + textePara);
 </script>
```



# Modifier le contenu d'un élément HTML

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On modifie le contenu textuel de notre élément h1 avec innerHTML
 document.getElementById('gros_titre').innerHTML = 'Titre modifié !';
 </script>
 </body>
</html>
```



## Titre modifié !

Du texte et [un lien](#)

Un deuxième paragraphe

# Modifier la valeur d'un attribut HTML

utiliser la propriété **attribute** de l'objet **Element**.

affecter une nouvelle valeur à l'attribut ciblé.

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On modifie la valeur de l'attribut href de notre lien
 document.querySelector('a').href = 'http://wikipedia.org';
 </script>
 </body>
</html>
```

# getAttribute et setAttribute

```
<body>

 Un lien modifié dynamiquement

 <script>

 var link = document.getElementById('myLink');

 var href = link.getAttribute('href'); // On récupère l'attribut « href »

 alert(href);

 link.setAttribute('href', 'http://www.monsite.com'); // On édite l'attribut « href »

 </script>

</body>
```

# Les attributs accessibles

```
<body>

Un lien modifié dynamiquement

<script>

 var link = document.getElementById('myLink');

var href = link.href;

 alert(href);

 link.href = 'http://www. monsite.com';

</script>

</body>
```

# ClassName

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On modifie la valeur de la class de
 document.querySelector('.para').class
 </script>
 </body>
</html>
```

The screenshot shows the Chrome DevTools interface with the DOM Tree panel open. The DOM structure is as follows:

- <!DOCTYPE html>
- <html>
- <head> ... </head>
- <body>
  - <h1 id="gros\_titre">Le DOM</h1>
  - <p class="paral">...</p>
  - <p class="para">Un deuxième paragraphe</p>
  - <script>...</script>
- </body>
- </html>

The element <p class="paral">...</p> is currently selected in the DOM tree. In the JavaScript console below, the command `document.querySelector('.para').class` is highlighted, indicating it has been run. The output of this command is also visible in the console.

---

# AJOUTER ET INSERER DES ELEMENTS

# Créer un nouvel élément

- Pour créer un nouvel élément HTML en JavaScript, nous utiliserons la méthode **createElement()** de l'objet **Document**.
- Nous avons ici créé un nouvel élément **p**. Cependant, notre élément ne contient **rien** pour le moment

```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On crée un élément de type p
 document.createElement('p');
 </script>
 </body>
</html>
```

# Ajouter un attribut et du texte à un élément

Ajout d'un attribut id avec la valeur « nouveau » à notre paragraphe.

Création d'un nœud de type texte que nous n'avons pas encore inséré

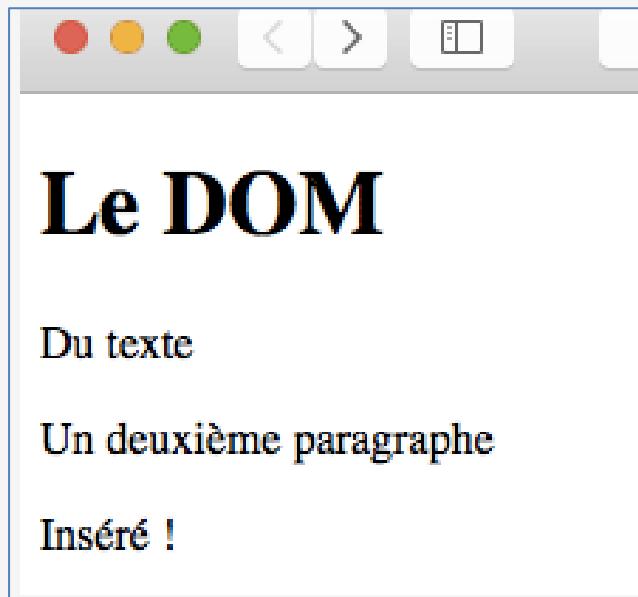
```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On crée un élément de type p
 var newPara = document.createElement('p');

 //On ajoute un attribut id à notre paragraphe
 newPara.id = 'nouveau';

 //On crée un noeud de type texte
 var texte = document.createTextNode('Inséré !');
 </script>
</body>
</html>
```

# Insérer dans la page HTML



```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On crée un élément de type p
 var newPara = document.createElement('p');

 //On ajoute un attribut id à notre paragraphe
 newPara.id = 'nouveau';

 //On crée un noeud de type texte
 var texte = document.createTextNode('Inséré !');

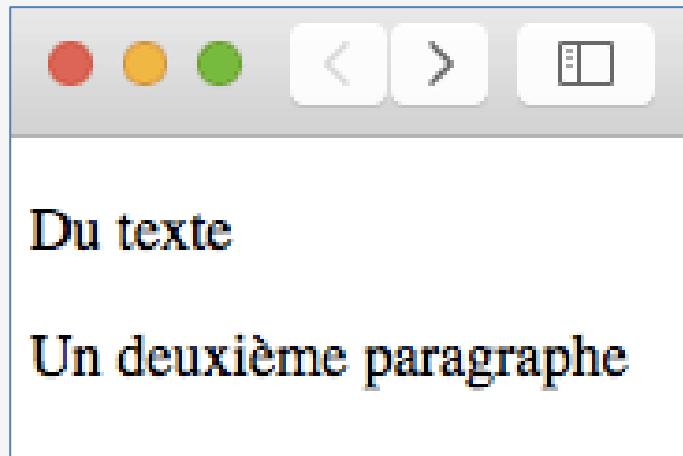
 //On insère le texte dans notre paragraphe
 newPara.appendChild(texte);

 /*On insère finalement notre élément en tant que
 *dernier enfant de body (auquel on accède directement
 avec "document.body", tout simplement)/
 document.body.appendChild(newPara);
 </script>
</body>
```



# Supprimer un élément

nous allons utiliser la méthode **removeChild()**.



```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 /*On veut supprimer notre titre, on commence donc
 par y accéder/
 var titre = document.getElementById('gros_titre');

 //On accède ensuite à l'élément parent de h1 (body)
 var parent = document.body;

 //On supprime finalement notre titre avec removeChild
 parent.removeChild(titre);
 </script>
 </body>
</html>
```

# Remplacer des éléments

Pour modifier ou remplacer des nœuds / éléments

HTML par d'autres, on va utiliser la méthode **replaceChild()**



```
<body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>

 <script>
 //On accède à notre élément h1 en JavaScript
 var titre = document.getElementById('gros_titre');

 //On accède ensuite à l'élément parent de h1 (body)
 var parent = document.body;

 //On crée une valeur de remplacement
 var nouveauTitre = document.createElement('h2');

 //On ajoute du texte et un id à notre h2
 nouveauTitre.id = 'titre_moyen';
 nouveauTitre.innerHTML = 'Titre modifié en Js !'

 //On remplace finalement h1 par h2
 parent.replaceChild(nouveauTitre, titre);
 </script>
</body>
```

# Parcourir le DOM

# La propriété parentNode



```
<!DOCTYPE html>
<html>
 <head>
 <title>Le DOM HTML</title>
 <meta charset="utf-8">
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>

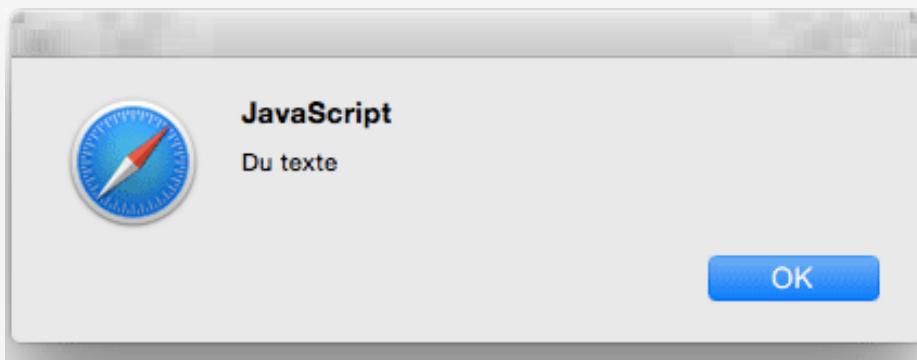
 <div>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>
 </div>

 <script>
 //On accède à notre premier élément p
 var p = document.querySelector('.para');

 //On accède ensuite à notre div avec parentNode
 var div = p.parentNode;

 //On peut ensuite manipuler notre div à loisir
 div.style.color = 'orange';
 </script>
 </body>
</html>
```

# childNodes et nodeValue



```
<body>
 <h1 id="gros_titre">Le DOM</h1>

 <div>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>
 </div>

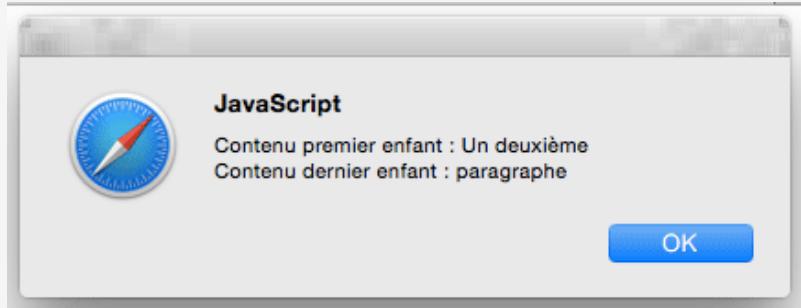
 <script>
 //On accède à notre body
 var b = document.body;

 /*On accède ensuite à notre div avec childNodes.
 *On se rappelle que le premier élément d'un tableau
 possède la clef 0, le second la clef 1, etc./
 var div = b.childNodes[3];

 //On accède ensuite au premier paragraphe dans notre div
 var p1 = div.childNodes[1];

 //Et enfin au texte dans notre paragraphe, qu'on affiche
 var texte = p1.childNodes[0].nodeValue;
 alert(texte);
 </script>
</body>
```

# firstChild et lastChild



```
<body>
 <h1 id="gros_titre">Le DOM</h1>

 <div>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>
 </div>

 <script>
 //On accède à notre deuxième paragraphe
 var p2 = document.querySelectorAll('.para')[1];

 //On accède aux premier et dernier enfants de p2
 var premier = p2.firstChild;
 var dernier = p2.lastChild;

 //On récupère et affiche ce qu'ils contiennent
 var inner1 = premier.nodeValue;
 var inner2 = dernier.innerHTML;

 alert('Contenu premier enfant : ' + inner1 +
 '\nContenu dernier enfant : ' + inner2);
 </script>
</body>
```

# nextSibling et previousSibling

d'accéder respectivement au nœud « frère » (de même niveau) **suivant** ou **précedent** le nœud ciblé.



```
<body>
 <h1 id="gros_titre">Le DOM</h1>

 <div>
 <p class="para">Du texte</p>
 <p class="para">Un deuxième paragraphe</p>
 </div>

 <script>
 //On accède à h1
 var titre = document.getElementById('gros_titre');

 /*On accède d'abord à l'espace entre notre h1 et notre div,
 puis à notre div avec le deuxième nextSibling/
 var div = titre.nextSibling.nextSibling;

 //On modifie le contenu textuel du div
 div.innerHTML = '<p>div totalement modifié !</p>'

 </script>
</body>
```

# Exercice : DOM : manipulation et création

On vous donne le fichier HTML suivant et on vous demande (amicalement!) de:

Compléter le code JavaScript **uniquement** permettant d'avoir comme résultat la page suivante ayant :

1. Un titre modifié,
2. Un lien hypertexte agrandis et colorié via les propriétés
3. Un paragraphe rajouté contenant le texte "*Un troisième paragraphe rajouté par JavaScript*"

```

<html>
 <head>
 <title>Le DOM HTML</title>
 </head>
 <body>
 <h1 id="gros_titre">Le DOM</h1>
 <p class="para">Du texte et un lien</p>
 <p class="para">Un deuxième paragraphe</p>
 <script>
 // TODO
 </script>
 </body>
</html>
```



---

# JSON et XML

# JSON vs XML

- JSON et XML peuvent être utilisés pour recevoir des données à partir d'un serveur Web.
- Les exemples JSON et XML suivants définissent tous deux un objet employé, avec un éventail de 3 employés :
- Pourquoi JSON est meilleur que XML:
  - XML est beaucoup plus difficile à parser que JSON
  - JSON est automatiquement traduit en un objet JavaScript prêt à l'emploi.

```
<employees>
 <employee>
 <firstName>John</firstName>
 <lastName>Doe</lastName>
 </employee>
 <employee>
 <firstName>Anna</firstName>
 <lastName>Smith</lastName>
 </employee>
 <employee>
 <firstName>Peter</firstName>
 <lastName>Jones</lastName>
 </employee>
</employees>
```

```
{"employees": [
 { "firstName":"John", "lastName":"Doe" },
 { "firstName":"Anna", "lastName":"Smith" },
 { "firstName":"Peter", "lastName":"Jones" }
]}
```

# JSON et JavaScript

## JavaScript Object Notation

- Un format de représentation des données

Très utilisé pour faire des échanges de données entre les API

- Transmettre des configurations,
- Echanger les données sur le Web,
- Enregistrer les avancées dans un jeux, etc.

○ <b>Strings</b>	“Hello World” “Kyle” “I”
○ <b>Numbers</b>	10 1.5 -30 1.2e10
○ <b>Booleans</b>	true false
○ <b>null</b>	null
○ <b>Arrays</b>	[1, 2, 3] [“Hello”, “World”]
○ <b>Objects</b>	{ “key”: “value” } { “age”: 30 }

Très léger et plus facile à comprendre que les notations XML qui ont trop de balises ouvrantes/fermantes

C'est un super-set de JavaScript qui offre le mêmes types de base de JavaScript

- Les Types sous JSON: String, numer, Boolean, null, Array, Object

```
user.json
{
 "key": "value",
 "key": "value"
}
```

Tout ce qu'on a faire, c'est de créer un **fichier.json** qui contient soit un objet ou un tableau à la racine pour servir de conteneur pour les reste des données

On peut bien sûr avoir des sous objet et des sous-tableaux dans notre structure JSON, par exemple:

- un objet nommé Personne,
- une liste d'amis sous la clé "friends" , etc.

Presque tout langage de programmation actuellement est capable de "**parser**" ses Objets dans le format JSON

Parsing des fichiers JSON sous JavaScript

- JSON accepte le fait d'avoir des valeurs différentes pour les mêmes clés
- Sous JavaScript, il n'est pas obligatoire d'avoir des doubles-cotes pour les nom de clés
- Sous JSON, elles sont obligatoires

```
user.json
{
 "name": "Kyle",
 "favoriteNumber": 3,
 "isProgrammer": true,
 "hobbies": ["Weight Lifting", "Bowling"],
 "friends": [
 {
 "name": "Joey",
 "favoriteNumber": 100,
 "isProgrammer": false,
 "friends": [...]
 }
}
```

Devinez les résultats

```
let formationString = JSON.stringify(formation);

console.log(JSON.parse(formationString)[0]);
console.log(JSON.parse(formationString)[1]["nom"]);
console.log(JSON.parse(formationString)[0]["nom"][0]);
```

```
let formation =
[
 {
 "nom": "Bassem Seddik",
 "nombreFormes": 7,
 "centre": "ORSYS",
 "qualité": 9.5
 },
 {
 "nom": "Autre formateur",
 "nombreFormes": 7,
 "centre": null,
 "qualité": 9
 }
];
```

# XML: Introduction

**XML: eXtensible Markup Language est**

- un **langage** informatique de **balisage générique**
- Technologie maintenant âgée de plus de 10 ans

## **langage de description**

- Permet de décrire et structurer un ensemble de données selon des règles et des contraintes prédéfinies
- Exemples
  - SGML, **XML** et HTML

## Application

- décrire l'ensemble des livres d'une bibliothèque,
- la liste des chansons d'un CD, etc

```
<?xml version="1.0"?>
<questionnaire>
 <question>
 Qui était le premier
 empereur romain ?
 </question>
 <réponse>
 Auguste
 </réponse>
 <!-- Note : tu auras besoin
 de plus de questions.-->
</questionnaire>
```

**XML**

# Composition d'un document XML

Un document XML commence par <?xml afin de préciser qu'il s'agit en effet d'un document XML

- Cette ligne terminera par ?>

Balises délimitées par les caractères "inférieur" '<' et "supérieur" '>'

- peuvent éventuellement contenir des attributs et/ou englober du texte libre

En HTML, toutes les balises sont définies. En XML il n'existe aucune balise prédéfinie

- c'est à nous de les définir ainsi que les attributs.

```
<?xml version="1.0"?>
<annuaire>
 <personne>
 <nom>ben flen1</nom>
 <prenom>flen1</prenom>
 </personne>
 <personne>
 <nom>ben flen2</nom>
 <prenom>flen2</prenom>
 </personne>
 <personne>
 <nom>ben flen3</nom>
 <prenom>flen3</prenom>
 </personne>
</annuaire>
```

# Ducument XML bien-formé

Il ne doit exister qu'une seule balise racine, son nom est laissé libre

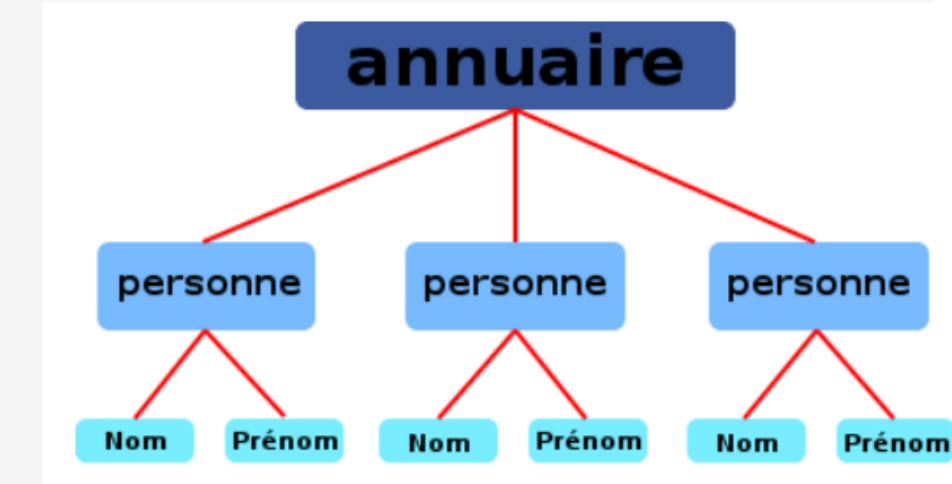
- si l'on choisi "**baliseracine**" comme balise racine
- alors l'ensemble des balises du document doit être compris entre <**baliseracine**> et </**baliseracine**>

Toute balise ouverte doit être fermée

- <**balise**> doit être associé </**balise**>
- Si une balise est vide (i.e. n'englobe pas de texte) alors elle pourra être simplifiée en une balise auto-fermante et s'écrira <**balise** />

Les balises doivent-être **imbriquées** les unes dans les autres

- une balise ouverte ne peut pas être fermée tant que toutes les balises incluses dedans n'ont pas été fermées



l'arbre correspondant au document donné en exemple

# Texte/attribut et commentaires

Ainsi nous pouvons compléter notre exemple précédent comme suit:

Le début de la section de commentaire doit être précisée par <!-- et la fin par -->

```
<?xml version="1.0"?>
<annuaire>
 <personne datenaissance="1942-01-08">
 <nom>ben flen1</nom>
 <prenom>flen1</prenom>
 </personne>
 <personne datenaissance="1952-01-09">
 <nom>ben flen2</nom>
 <prenom>flen2</prenom>
 </personne>
 <personne datenaissance="1962-01-10">
 <nom>ben flen3</nom>
 <prenom>flen3</prenom>
 </personne>
</annuaire>
```

```
<?xml version="1.0"?>
<!-- Mon document version 1.0-->
<informatique>
 <!-- Section XML -->
 <cours>...</cours>
 <!-- Section PHP -->
 <cours>...</cours>
</informatique>
```

# Document Valide

Si l'on fait la comparaison avec l'HTML, on sait que:

```
<html>
 <head>
 </head>
 <body>
 <h1>Mon site web</h1>
 </body>
</html>
```

**Correcte**

```
<html>
 <head>
 </head>
 <h1>Mon site web</h1>
</html>
```

**Non Correcte**

Il y a certaines règles à respecter même si les navigateurs ont été programmés pour comprendre toutes chose désirée:

- la balise `<html>` doit contenir les balises `<head>` et `<body>`.
- La balise `<h1>` ne peut apparaître directement après `<head>`

**Par analogie:** afin d'être traité correctement un fichier XML doit répondre à une structure donnée afin retrouver les informations. On peut attendre:

- telle ou telle information à telle endroit,
  - telle ou telle type de valeur dans tel ou tel attribut,
- donc le document XML doit être **valide**

# Vérifier la validité d'un document XML

- Permet de définir une ensemble de règles explicites dans un document à part
- Sert au contrôle automatique du document XML par un outil approprié

## document XML

Il existe 2 normes de documents de validation:

- DTD : Document Type Definition**
- XML Schema

Pour définir une DTD externe, il suffit de:

- écrire **<!DOCTYPE racine SYSTEM "nomdufichier.dtd">**

```
<?xml version="1.0"?>
<!DOCTYPE encyclopedie SYSTEM "encyclopedie.dtd">
<encyclopedie>
 <personne datenaissance="1942-01-08" genre="H">
 <nom>Ben flen1</nom>
 <prenom>flen 1</prenom>
 <publication>Une breve histoire du temps</publication>
 </personne>
 <personne datenaissance="1932-07-13" genre="H">
 <nom>ben flen2</nom>
 <prenom>flen2</prenom>
 <publication>L'Univers explique a mes petits-enfants</publication>
 <publication>Poussieres d'etoiles</publication>
 </personne>
 <personne datenaissance="1879-03-14" genre="F">
 <nom> ben flen 3</nom>
 <prenom>flen 3</prenom>
 <publication>Des ondes gravitationnelles</publication>
 <publication>Sur la theorie quantique du rayonnement</publication>
 </personne>
</encyclopedie>
```

## DTD correspondant

```
<?xml version="1.0"?>
<!ELEMENT encyclopedie (personne*)>
<!ELEMENT personne (nom,prenom,publication+)>
<!ATTLIST personne genre (H | F) "H">
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT publication (#PCDATA)>
```

# DTD, comprehension des champs

- Pour chacun des éléments, on définit sa composition par **<!ELEMENT nom\_element (structure)>**
- Pour chacun des attributs, on définit sa composition par **<!ATTLIST nom\_element nom\_attribut (structure)>**
- **personne\*** signifie que dans un élément `<encyclopedie>` on peut trouver **de 0 à plusieurs éléments** de type `<personne>` à l'exclusion de tout autre élément
- **nom,prenom,publication+** signifie que dans un élément **personne** on doit trouver
  - un et un seul élément `<nom>`,
  - un et un seul élément `<prenom>`
  - et **un ou plusieurs (+)** éléments `<publication>` et dans cet ordre (par la virgule)

```
<?xml version="1.0"?>
<!ELEMENT encyclopedie (personne*)>
<!ELEMENT personne (nom,prenom,publication+)>
<!ATTLIST personne genre (H | F) "H">
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT publication (#PCDATA)>
```

- L'attribut genre de la balise **<personne>** est, ici, défini comme étant un choix imposé entre "**H**" et "**F**"
  - Les valeurs possibles sont séparées par des caractères '|'
  - Cet attribut n'étant pas indiqué comme obligatoire, s'il n'est pas précisé alors il prendra par défaut la valeur "H"
- `<nom>`, `<prenom>` et `<publication>` sont des éléments qui peuvent contenir du texte comme l'indique #PCDATA.

# Programme de formation

I- Les technologies du Web

II- Le langage JavaScript, prise en mains

III- Evénements et données

IV- Gestion de formulaires HTML

V- Interaction avec les CSS

VI- Manipulation du DOM XML

VII- Ajax

## VII- Ajax

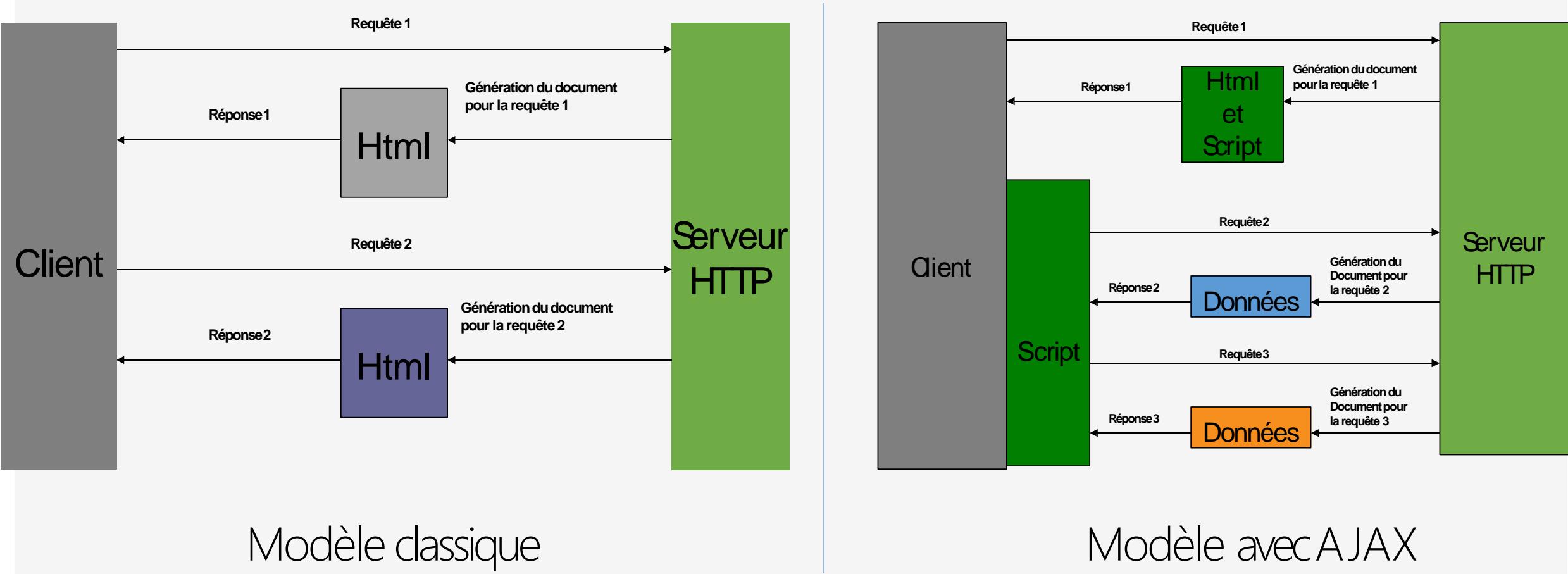
1. Présentation de Asynchronous JavaScript And Xml.
2. Enjeux, solutions et alternatives.
3. Les bibliothèques du marché.
4. HTTP et Ajax : échanges HTTP et l'objet XMLHttpRequest.

**Exercice:** *Récupération et affichage de données XML. Interrogation d'une base de données.*

*Création d'un formulaire de connexion.*

- **AJAX (Asynchronous JavaScript and XML)** est une technique pour créer des pages web rapides et dynamiques.
  - Lire les données d'un serveur Web - après que la page a chargé
  - Mettre à jour une page Web sans recharger la page
  - Envoyer/recevoir des données à un serveur Web - en arrière-plan
- AJAX permet de mettre à jour les pages web de manière **asynchrone** par l'échange de petites quantités de données avec le serveur (en arrière plan). Cela veut dire qu'il est possible de mettre à jour des parties d'une page web, sans avoir à recharger toute la page.
- Les pages web classiques, (n'utilisant pas AJAX) doivent recharger la page entière si le contenu doit changer.
- Exemples d'applications utilisant AJAX: Google Maps, Gmail, YouTube et Facebook.

# Comparaison des échanges sans et avec AJAX



# Principes de AJAX

- AJAX est basé sur les standards d'internet, et utilise une combinaison de :
  - L'objet **XMLHttpRequest** (pour récupérer des données à partir d'un serveur Web)
  - **JavaScript/DOM** (pour afficher/utiliser les données)
- AJAX utilise un modèle de programmation comprenant d'une part la **présentation**, d'autre part les **événements**
  - Les évènements sont les actions (souvent de l'utilisateur), qui provoquent l'appel des fonctions associées aux éléments de la page
  - Ces fonctions JavaScript identifient les éléments de la page grâce au **DOM** et communiquent avec le serveur par l'objet **XMLHttpRequest**

# XMLHttpRequest

- Tous les navigateurs modernes (Chrome, IE7 +, Firefox, Safari, et Opera) ont maintenant l'objet intégré **XMLHttpRequest**.
- L'objet **XMLHttpRequest** est utilisé pour échanger des données avec un serveur (en arrière plan de manière asynchrone).
- Cela veut dire qu'il est possible de mettre à jour des parties d'une page web, sans avoir à recharger toute la page.
- La syntaxe pour créer un objet XMLHttpRequest :

```
let xhttp = new XMLHttpRequest();
```

# AJAX - Envoyez une requête à un serveur

Pour envoyer une demande à un serveur, il faut utiliser les méthodes `open()` et `send()` de l'objet `XMLHttpRequest`:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Method	Description
<code>open(<i>method</i>, <i>url</i>, <i>async</i>)</code>	Specifies the type of request  <i>method</i> : the type of request: GET or POST <i>url</i> : the server (file) location <i>async</i> : true (asynchronous) or false (synchronous)
<code>send()</code>	Sends the request to the server (used for GET)
<code>send(<i>string</i>)</code>	Sends the request to the server (used for POST)

# AJAX – Réponse du serveur

- Pour obtenir la réponse d'un serveur, utiliser la propriété `responseText` ou `responseXML` de l'objet `XMLHttpRequest`.
  - `responseText` : obtenir la réponse sous forme de chaîne
  - `responseXML` : obtenir la réponse en format XML

```
document.getElementById("demo").innerHTML =
 xhttp.responseText;
```

# AJAX: onreadystatechange

- Quand la requête envoyée vers serveur, nous allons effectuer certaines actions en fonction de la réponse.
  - L'événement `onreadystatechange` est déclenché chaque fois la propriété `readyState` change.
  - La propriété `readyState` détient le statut de l'objet `XMLHttpRequest`.
- Trois propriétés importantes de l'objet XMLHttpRequest existent:

Property	Description
<code>onreadystatechange</code>	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
<code>readyState</code>	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
<code>status</code>	200: "OK" 404: Page not found

# AJAX: Exemple complet

- Noter que l'appel à la réponse est conditionné par un changement réussi de l'état de la page

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8" />
</head>
<body>
 <div>
 <h2 id="demo">Ce texte sera remplacé par le text récupéré via AJAX</h2>
 <button type="button" onclick="loadDoc()">Charger nouveau contenu</button>
 </div>
 <script>
 function loadDoc() {
 var xhttp = new XMLHttpRequest(); xhttp.onreadystatechange = function () {
 if (this.readyState == 4 &&this.status == 200) {
 document.getElementById("demo").innerHTML = this.responseText;
 }
 };
 xhttp.open("GET", "info.txt", true);
 xhttp.send();
 }
 </script>
</body>
</html>
```

**Ce texte sera remplacé par le text récupéré via AJAX**

Charger nouveau contenu

# JSON et son usage sous AJAX

Rappel:

- JSON (JavaScript Object Notation) est une syntaxe pour stocker et échanger des données
  - est une alternative plus facile à utiliser au format XML
  - est un format de données d'échange léger
  - est "auto-descriptif" et facile à comprendre
  - Le format JSON est syntaxiquement identique au code pour créer des objets JavaScript.

```
{"employees": [
 {"firstName": "John", "lastName": "Doe"},
 {"firstName": "Anna", "lastName": "Smith"},
 {"firstName": "Peter", "lastName": "Jones"}
]}
```

En raison de cette similitude, **au lieu d'utiliser un parseur** (comme XML fait), un programme JavaScript peut **utiliser les fonctions standard JavaScript** pour convertir les données JSON en objets JavaScript natifs.

# JSON VS XML

- Pour les applications AJAX, JSON est plus rapide et plus facile que XML:

## XML

- Récupère un document XML
- Utilise le DOM XML pour convertir le document XML
- Extrait les valeurs et les stocker dans des variables

## JSON

- Récupère une chaîne JSON
- `JSON.Parse` pour analyser la chaîne JSON

# JSON : Le 'nouvel' X dans AJAX

- Les données JSON sont intégrées dans la page:

```
<html>...
 <script>
 var data = { ... JSONdata ... };
 </script>
</html>
```

- Etapes:

1. Envoyer un objet JSON
2. Obtenir **responseText**
3. Parser **responseText**

```
var req = new XMLHttpRequest();
req.open("GET", "fichier.json", true);
req.send();
```

```
// Permet de transformer la chaîne reçue en objet JSON
responseData = JSON.parse(responseText);

// Permet d'évaluer une expression JS envoyée sous forme chaîne
responseData = eval('('+req.responseText+')');
```

# JSONP

- JSON fonctionne bien avec **XMLHttpRequest**, mais pour des raisons de sécurité, les navigateurs interdisent de faire du « cross-domain » avec XMLHttpRequest.
  - Il n'est pas possible d'utiliser JSON pour des demandes à d'autres sites.
- **JSONP (JSON with Padding)** est une technique utilisée pour surmonter les restrictions inter-domaines imposées par les navigateurs pour permettre aux données d'être récupérées à partir de serveurs autres que celui d'où la page a été servie.

## Principe de JSONP

- Au lieu d'aller charger un fichier JSON, on **charge un script** : une fonction JavaScript de **callback**
- La fonction est renvoyée par le serveur au client qui l'exécute. Il est alors autorisé à charger des données JSON depuis un domaine externe, et donc à notifier l'objet appelant avec cette fonction de callback.
- **JSON/JSONP** est proposé de la sorte par de nombreux services en ligne, telles que l'API Flickr, par exemple.

# JSONP de la librairie jQuery

- JSONP est une méthode d'envoi de données JSON sans se soucier des problèmes cross-domaines.
  - **JSONP n'utilise pas l'objet XMLHttpRequest.**
  - JSONP utilise le tag **<script>** à la place pour envoyer du contenu JSON.
    - C'est plus léger, plus rapide et profite des méthodes prédisposées sous JS pour manipuler les fichiers JSON
    - L'usage de la balise **<script>** est toléré par les serveurs, alors que **XMLHttpRequest** fait face à des limitations
- La demande d'un fichier à partir d'un autre domaine peut causer des problèmes, en raison de la stratégie de sécurité inter-domaines (**cross-domain policy**).
  - Demander un script externe à partir d'un autre domaine n'a pas ce problème.
  - JSONP utilise cet avantage, et demande des fichiers en utilisant la balise de **script** au lieu de l'objet XMLHttpRequest.

# JSONP: Exemple

- La première étape est d'encapsuler les données - ici des statistiques - dans une fonction de callback, dont on choisit le nom, par exemple `jsoncallback()`.

```
<script>
 function myFunc(myObj) {
 document.getElementById("demo").innerHTML =
 myObj.name;
 }
</script>
<script src="demo_jsonp.php"></script>
```

- Cette chaîne de texte sera retornée par une URL en HTTP, par exemple :

[http://localhost/demo\\_jsonp.php?callback=?](http://localhost/demo_jsonp.php?callback=?)

- Par convention, on se sert du dernier argument de l'URL (le `?` ) pour placer le nom de la fonction de retour **jsoncallback**
- Il suffit sur la page destinée à recevoir les informations et de traiter les données reçues.

Exemple d'appel à un script local sous jQuery (le `$.` est juste un alias à la classe JQUERY)

- Présent sur le serveur de l'API `/api/getWeather`
- avec le paramètre de requête `zipcode = 97201`
- Et en cas de succès de la demande, on remplace le html de l'élément `#weather-temp` dans notre page par le texte renvoyé.

```
$.ajax({
 url: "/api/getWeather",
 data: {
 zipcode: 97201
 },
 success: function(result) {
 $("#weather-temp").html("" + result + " degrees");
 }
});
```

# Web APIs et compatibilité inter navigateurs

- Une API Web (Application Programming Interface) est le rêve de tout développeur !
  - Elle peut étendre les fonctionnalités du navigateur
  - Elle simplifie énormément les fonctions complexes
  - Elle peut fournir une syntaxe facile au lieu de codes complexe
- Tous les navigateurs ont un ensemble d'API Web intégrées pour prendre en charge des opérations complexes et pour aider à accéder aux données.
- Par exemple, l'API de géolocalisation peut renvoyer les coordonnées de l'emplacement du navigateur.

## AJAX: points à retenir

- ✓ AJAX permet de récupérer des données d'un serveur tout en restant sur la même page
- ✓ L'objet **XMLHttpRequest** permet de gérer l'interactivité avec le serveur
- ✓ Les données récupérés sont souvent au format JSON



Téléchargez les exemples de codes présents sur l'espace GitHub relatif:



[https://github.com/bassemSeddik/INETUM\\_JavaScript](https://github.com/bassemSeddik/INETUM_JavaScript)



contact formateur: [bassem.seddk@sousse.r-iset.tn](mailto:bassem.seddk@sousse.r-iset.tn)

## Références

- Thierry Templier, Arnaud Gougeon, "Javascript pour le web 2.0, Programmation objet, dom, ajax, prototype, dojo, script.aculo.us, rialto...", Eyrolles, 2007
- Ramzi Farhat, "JavaScript, HTML dynamique", ORSYS, 2020
- Fabien Henon, OpenClassRoom, Écrivez du JavaScript pour le web, 2021
- Pierre Giraud, "Apprendre à coder en JavaScript | Cours complet" , 2020
- Apprenez à programmer avec JavaScript, Will Alexander, Openclassrooms, 2020
- JavaScript and the DOM: Dynamically control the browser, Udacity course, 2019
- Intro to JavaScript, Udacity course, 2019
- Object-Oriented JavaScript, Udacity course, 2019
- Référence JavaScript, Mozilla MDN web Docs, 2020

# Projet

Voir exercice de codage: Solution pour la gestion des transaction de revenus et de dépenses

/INETUM JavaScript (partie projet)