

JavaScript

INTRODUCTION

Historique

- Créé en 1995 par (et pour) Netscape
- Initialement appelé LiveScript
- Standardisé par ECMA en 1997
- Version actuelle : 11 Version ES2020

Historique

1997	1998	1999	!publié	2009	2011
ES1	ES2	ES3	ES4	ES5	ES5.1
		Expressions régulières try/catch	Jamais publié	Strict mode JSON String.trim() Array.isArray() Itération dans les tableaux	
				Est supporté par tous les navigateurs modernes	

Historique

2015	2016	2017	2018
ES6 ou ECMAScript 2015	ES7 ou ECMAScript 2016	ECMAScript 2017	ECMAScript 2018
Let & const Paramètre avec valeurs par défaut Array.find() Array.findIndex() Fonctions fléchées	Exponentiel(**) Array.prototype.includes	String.padStart() Nouvelles propriétés sur des objet Fonctions asynchrones Partages de mémoire	Propriétés rest/spred Itération asynchrone Promise.finally() RegExp
Supporté par la plupart des navigateurs sauf Internet Explorer	Supporté par Chrome et Opera uniquement		

Historique

2019

ES10 ou ECMAScript 2019

Array.Flat() : Créer un tableau à partir de tableau de tableau

Array.flatMap() : Idem .map mais le tableau n'a qu'une dimension

Object.fromEntries(), String.trimStart() & String.trimEnd(), Optional Catch Binding, Function.toString(), Symbol.description, Well Formed JSON.Stringify(), Array.Sort Stability, JSON \subset ECMAScript (JSON Superset)

Plus d'informations :

<https://medium.com/@selvaganesh93/javascript-whats-new-in-ecmascript-2019-es2019-es10-35210c6e7f4b>

Supporté par la plupart des navigateurs sauf Internet Explorer

Historique

2020 (16 octobre 2019)

ES2020 ou ECMAScript 2020

1 nouveau type primitif = BigInt (Stocker et faire des opérations de manière plus précise sur de gros entiers.

Plus d'informations :

<https://tc39.es/ecma262/>

JavaScript

- JavaScript
 - Langage interprété par le navigateur qui a pour but de dynamiser les sites Internet
 - Langage orienté prototype (pas de classe comme en C#)
- ECMAScript
 - Langage de programmation standardisé dont les spécifications sont mises en œuvre dans le JavaScript.
- JavaScript vs. Java

JavaScript vs. PHP

Caractéristique	PHP	Javascript
Exécution	Exécuté sur le serveur.	Exécuté chez le client.
Nécessaire à l'exécution	Un interpréteur PHP doit être installé sur le serveur.	Tous les navigateurs possèdent un interpréteur Javascript (mais peut être désactivé).
Manipulation de fichiers	Lecture, écriture, ajout possible dans des fichiers texte et éventuellement binaire situés sur le serveur.	Totalement incapable de manipuler les fichiers.
Manipulation de BD	PHP permet d'interroger tout type de base de donnée	Impossible en Javascript.
Richesse	Plus de 2000 fonctions.	Tout au plus une centaine de fonction.
Information sur le serveur	Multitude d'informations concernant le serveur.	Impossible en Javascript.
Information sur le système du visiteur	En dehors du nom du système d'exploitation du visiteur, on ne peut rien obtenir.	En Javascript, il est possible d'établir la résolution de l'écran, ainsi que les plugins ... de l'utilisateur.
Réagir aux événements chez le client	Impossible en PHP, puisqu'il est exécuté côté serveur.	Javascript permet de réagir aux événements : (dé)chargement d'une page, validation de formulaire, clic...

Exemple JavaScript

The screenshot displays the Gmail web interface. At the top, there's a navigation bar with links to Gmail, Calendar, Documents, Photos, Reader, Web, and more. A search bar is prominently featured with a 'SEARCH MAIL' button and a 'SEARCH THE WEB' button. Below the navigation bar, the left sidebar shows the 'Mail' section with a 'COMPOSE MAIL' button and a list of folders: Inbox (2), Starred, Sent Mail, Drafts (2), Hiking (3), Urgent!, and 12 more. The 'Hiking' folder is selected. The main content area shows two email conversations. The first conversation is from 'Kenneth Joel' to 'me', dated Mar 24. The email text reads: 'Hey there! Long time no speak. How have things been? Do you want to meet up for lunch tomorrow? Hope to hear from you soon. Kenneth'. The second conversation is from 'Hiking Fan' to 'Kenneth', also dated Mar 24. The email text reads: 'Hey Kent! Things have been really good! And lunch sounds great. Want to go to our usual spot? Let me know! :) Bob'. The right sidebar shows the contact 'Kenneth Joel' with an email address 'bakescakes@gmail.com' and a 'Show details' link. Below this, there are several advertisements, including 'Holistic Nutrition School', 'Tojokan Shinkendo Dojo', 'V8 V-Fusion® + Tea', and 'Brain Test™'. The bottom of the interface shows a 'Chat' section with a search bar and a list of contacts: Hiking Fan, Call phone, Arielle, Emily, Michael, Paul, and aalaloue.

Exemple JavaScript

The screenshot shows a Google Sheet titled "Movies_TechCrunch". The sheet contains a list of movies in column A, and their release dates and box office earnings in columns B and C respectively. The "Data" menu is open, showing options like "Sort sheet by column C, A → Z", "Sort sheet by column C, Z → A", "Sort range...", "Named and protected ranges...", "Filter", "Filter views...", "Pivot table report...", and "Validation...". The "Filter" option is selected, and a sub-menu is open showing "Create new filter view", "Filter view options", "Best movies (IMDB)", "Highest earning", and "Best movies (Rotten Tomatoes)".

	A	B	C	D	E	F	G
1	Title						
2	A Hard Day's Night						
3	Aliens						
4	Annie Get Your Gun						
5	Before Sunrise						
6	Cat on a Hot Tin Roof						
7	Henry V	11/8/1989	100				
8	Iraq for Sale: The War Profiteers	9/8/2006	100				
9	Jaws	6/20/1975	100				
10	Krrish	6/23/2006	100				
11	Love and Death	6/10/1975	100				
12	Mary Poppins	8/26/1964	100				

IMDB Rating

IMDB Votes

7.6

15,291

7.5

84

6.1

2,735

7.6

12,111

7.7

34,302

Exemple JavaScript



Exemples d'utilisation du JavaScript

- Exemple de base :

<http://jsbin.com/dokogetebe/2/edit>

- Animations en JavaScript :

<http://mrdoob.com/projects/chromeexperiments/google-gravity/>

<http://hereistoday.com/>

- De la 3D en JavaScript :

http://shapejs.shapeways.com/creator/?li=devhome_main

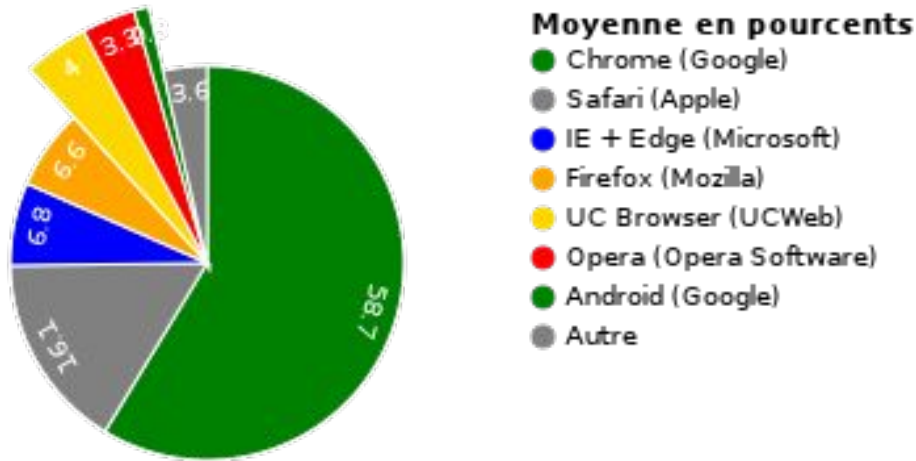
http://mrdoob.github.io/three.js/examples/webgl_materials_cars.html

Les limites du JavaScript

- Compatibilité entre navigateurs
 - Les différences sont de moins en moins importantes
- JavaScript est un langage exécuté côté client
 - Presque plus aucun internaute bloque l'exécution du JS

JavaScript et les navigateurs

Nécessité de vérifier la compatibilité des navigateurs



Sources, ressources et outils

- Ressources :
 - [Mozilla Developer Network](#)
 - [W3Schools](#)
 - [Developpez.com](#)
 - [OpenClassRooms](#)
- Outils :
 - JSBin : <http://jsbin.com>
 - JSFiddle : <http://jsfiddle.net>
 - Rubular : <http://rubular.com>
 - Chrome : Console Debug (F12)

ÉCRIRE SON CODE

Écrire du JavaScript

Tout comme pour le HTML et CSS, il ne faut rien de particulier pour écrire du JavaScript.

Un simple éditeur de texte suffit !

Le code est exécuté directement par le navigateur !

Écrire du JavaScript

Cependant, il existe des éditeurs de texte proposant l'auto complétion pour ce langage

- Eclipse (gratuit)
- Aptana Studio (gratuit)
- NetBeans (gratuit)
- Free JavaScript Editor (gratuit)
- WebStorm (payant)

Écrire du JavaScript

- Plugin pour Sublime Text :
 - SublimeCodeIntel : <http://sublimecodeintel.github.io/SublimeCodeIntel/>
- Plugin pour Notepad++ :
 - Pas de plugin, l'auto complétion est là, même si elle n'est pas parfaite.

Les commentaires

- Le JavaScript est un langage qui peut vite être incompréhensible.
- Il est très important de mettre des commentaires au sein de son code !

- Commentaire sur une ligne : `// Mon commentaire`

- Commentaire en bloc :
`/* Mon commentaire
sur plusieurs
lignes */`

Déboguer son code

La plupart des navigateurs offrent une console JavaScript.

Celle-ci est utile pour afficher les messages d'erreur ou les messages de débogage.

Déboguer son code

Pour afficher un message ou une variable dans la console :

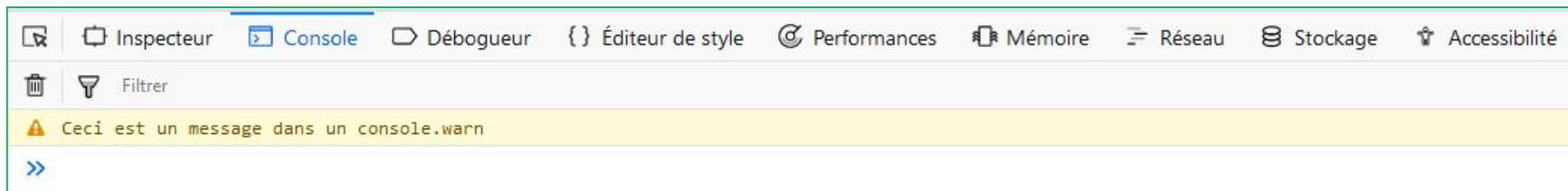
```
console.log(...);
```



Déboguer son code

Pour afficher un message ou une variable dans la console :

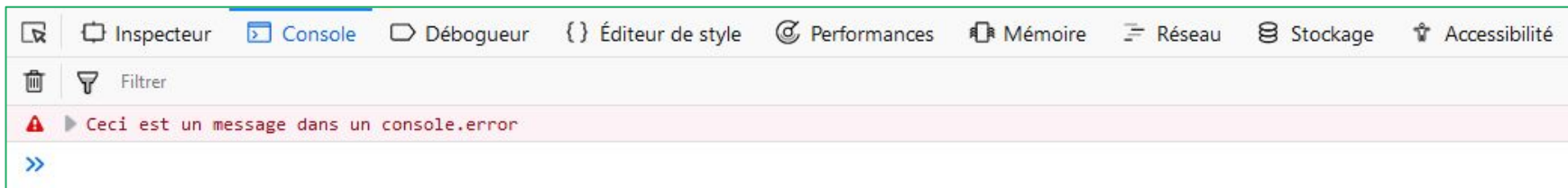
```
console.warn(...);
```



Déboguer son code

Pour afficher un message ou une variable dans la console :

```
console.error(...);
```



EMPLACEMENT DU CODE

Emplacement du code

Dans une page HTML:

```
<script type="text/javascript">  
    var maVariable;  
    ...  
</script>
```

Dans un fichier externe:

```
<script type="text/javascript" src="js/script.js"></script>
```

Dans un attribut événement - **A éviter!**

```
<a onclick="var maVariable; ...">Accueil</a>
```

Emplacement du code

Remarque: le code JS est exécuté par le navigateur de manière séquentielle à la lecture de la page web.

Attention à l'emplacement du code.

Emplacement du code

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8 />
  <title>Titre</title>
</head>

<body>
<div id="01">Bienvenue</div>

<script type="text/javascript">
el = document.getElementById("01");
el.innerHTML = "Bonjour";
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8 />
  <title>Titre</title>
</head>

<body>
<script type="text/javascript">
el = document.getElementById("01");
el.innerHTML = "Bonjour";
</script>

  <div id="01">Bienvenue</div>
</body>
</html>
```

La balise <noscript>

- La balise <noscript>...</noscript> permet d'afficher un texte au cas où le navigateur ne supporte pas le JavaScript, ou si celui-ci a été désactivé.

```
<noscript>Votre navigateur ne supporte pas le JavaScript ou le  
JavaScript a été désactivé.</noscript>
```

Un premier exercice

1. Créez une page HTML contenant le script JS suivant :
`alert("Bienvenue sur mon site");`
2. Migrez ce script dans un fichier "script.js" et faites-y appel dans votre page HTML

CONSTANTES ET VARIABLES

Constantes prédéfinies : NaN

Not a Number, indique que l'élément concerné n'est pas un nombre valide.

Cette valeur est renvoyé quand une fonction mathématique échoue `Math.sqrt(-1)` ou lorsqu'une conversion échoue `parseInt("blabla")`

Ne jamais tester la valeur NaN avec une égalité stricte:

```
NaN === NaN;           // false
```

```
Number.NaN === NaN;    // false
```

Toujours utiliser la méthode `isNaN(...)`:

```
isNaN(NaN);             // true
```

```
isNaN(Number.NaN);      // true
```

Constantes prédéfinies : Infinity

Infinity indique que l'élément concerné est hors des plages de valeurs définies pour un nombre

Infinity se comporte comme l'infini mathématique:

`Infinity * Infinity = Infinity`

`500 / Infinity = 0`

`500 / 0 = Infinity`

`Infinity/Infinity = NaN`

Constantes prédéfinies : Undefined

Indéfini, l'élément ne correspond à aucun type reconnu

Tester Undefined:

```
var x;  
x == undefined           // true  
typeof x == 'undefined'  // true  
  
var y = 8;  
y == undefined           // false  
typeof y == 'undefined'  // false
```

Constantes prédéfinies

- Il existe d'autres constantes reliées à des Objets Javascripts :
 - Math
 - Number

Constantes de Math

- **Math.PI** = valeur de pi avec 15 décimales

```
alert(Math.PI); // Affiche 3.14159...
```

- **Math.SQRT2** = Valeur de $\sqrt{2}$

```
alert(Math.SQRT2); // Affiche  
1.414...
```

Constantes de number

- Number.MAX_VALUE

La plus grande valeur possible en JavaScript : 1.79×10^{308}

- Number.MIN_VALUE

La plus petite valeur positive en JavaScript : 5.00×10^{-324}

Constantes définies par l'utilisateur

Permet de créer une constante nommée, accessible uniquement en lecture.

Il est nécessaire d'initialiser une constante lors de sa déclaration.

```
const NAME1 = value1;
```

Constantes définies par l'utilisateur

Nom de variable en MAJUSCULE et les mots séparés par un underscore

```
const MA_CONSTANTE = "blablabla";
```


Variables et typage

- JavaScript est un langage à typage dynamique
- Le type d'une variable est défini au runtime et peut être changé en cours d'exécution
- Nom des variables
 - Commence par une lettre, un underscore ou un dollar
 - Les caractères suivants sont des alphanumériques, underscores ou dollars

```
let variable = "mon texte d'initialisation";
```

```
(...)
```

```
variable = 2;
```

Variables - Formes littérales

//Forme littérale pour un nombre entier en base décimale

```
let nombreEntier = 11;
```

//Forme littérale pour un nombre réel

```
let nombreReel = 11.435;
```

//Forme littérale d'une chaîne de caractères

```
let chaineCaracteres = "Une chaîne de caractères";
```

//Forme littérale d'un tableau normal(object)

```
let tableau = [ "Premier élément", "Second élément" ];
```

//Forme littérale d'un tableau associatif/objet

```
let tableauAssociatif = { "cle1" : "valeur1", "cle2" : "valeur2" };
```

Où se trouvent les erreurs ?

```
let mavariable = 23;  
let 3fois9 = 27;  
let troisfois9 = 27;  
let le total = 100;  
let @mail = "jean.dupont@gmail.com";  
let variable = 30+20+10;  
let phrase = Bonjour tout le monde !;  
let age = "26";  
let resultat = 3 + 3 + "3";
```

Comment améliorer le code en respectant les bonnes pratiques ?

Variables - Portée des variables

Attention au lieu de déclaration d'une variable :

```
var maVariable = "Hello";
```

Accessible partout dans le code (même dans la fonction)

```
function maFonction(){  
    var maVariable2 = "World";  
    ... Code ...  
}
```

Accessible uniquement dans la fonction.

Variables - Portée des variables

Attention au lieu de déclaration d'une variable :

```
let maVariable = "Hello";
```

Accessible partout dans le code (même dans la fonction)

```
function maFonction(){  
  let maVariable2 = "World";  
  ... Code ...  
}
```

Accessible uniquement dans la fonction et à l'intérieur de ses accolades parentes.

```
for(let i = 0; i < 4; i++){  
  console.log(i);    //OK  
}  
console.log(i);      //KO
```

Variables - Portée des variables

Attention au lieu de déclaration d'une variable :

```
maVariable = "Hello";  
function maFonction(){  
    maVariable2 = "World";  
    ... Code ...  
}  
maFonction();  
console.log(maVariable2);    //OK  
  
for(i = 0; i < 4; i++){  
    console.log(i);    //OK  
}  
console.log(i);    //OK
```

Accessible partout dans le code (même dans la fonction)

Accessible partout (une fois la fonction exécutée).

Attention qu'une variable globale créée à l'extérieur d'une fonction et accessible dans la fonction (donc modifiable)

Variables – Que choisir entre var, let, const ?

On évite d'utiliser le var.

On utilise le **const** pour toute "variable" qui ne sera pas réaffecté.

On utilise le **let** pour toute variable qui sera réaffectée (pour un compteur dans une boucle par exemple, une chaîne de caractères).

Variables et typage - Types primitifs

En JavaScript, les types primitifs sont des "pseudo-objets" qui possèdent des propriétés et des méthodes

Boolean	Type dont les valeurs possibles sont true et false.
Null	Type dont l'unique valeur possible est null. Une variable possède cette valeur afin de spécifier qu'elle a bien été initialisée mais qu'elle ne pointe sur aucun objet.
Number	Type qui représente un nombre.
String	Type qui représente une chaîne de caractères
Undefined	Type dont l'unique valeur possible est undefined. Une variable définie possède cette valeur avant qu'elle soit initialisée.
Object	Type instancié grâce au mot clé new

Variables et typage

```
let variable1;  
alert(" type de variable1 : "+(typeof variable1));  
// variable1 est de type undefined  
  
let variable2 = null;  
alert(" type de variable2 : "+(typeof variable2));  
// variable2 est de type object  
  
let variable3 = 12;  
alert(" type de variable3 : "+(typeof variable3));  
// variable3 est de type number  
  
let variable4 = "une chaîne de caractères";  
alert(" type de variable4 : "+(typeof variable4));  
// variable4 est de type string  
  
let variable5 = true;  
alert(" type de variable5 : "+(typeof variable5));  
// variable5 est de type boolean
```

Variables et typage

JavaScript supporte les conversions de types primitifs en chaînes de caractères.

```
let booleen = true;  
let variable1 = booleen.toString();  
// variable1 contient la chaîne de caractères « true »
```

```
let nombreEntier = 10;  
let variable2 = nombreEntier.toString();  
// variable2 contient la chaîne de caractères « 10 »
```

```
let nombreReel = 10.5;  
let variable3 = nombreReel.toString();  
// variable3 contient la chaîne de caractères « 10.5 »
```

Variables et typage

JavaScript supporte les conversions de types primitifs en chaînes de caractères en spécifiant la base

```
let nombreEntier = 15;  
let variable1 = nombreEntier.toString();  
// variable1 contient la chaîne de caractère « 15 »
```

```
let variable2 = nombreEntier.toString(2);  
// variable2 contient la chaîne de caractère « 1111 » (binaire)
```

```
let variable4 = nombreEntier.toString(16);  
// variable4 contient la chaîne de caractère « f » (hexadécimal)
```

Variables et typage

JavaScript supporte les conversions de types chaînes de caractères en nombres entiers ou réels.

```
let entier1 = parseInt("15");  
// entier1 contient le nombre 15
```

```
let entier2 = parseInt("f", 16);  
// entier2 contient le nombre 15
```

```
let reel = parseFloat("15.5");  
// reel contient le nombre réel 15,5
```

Détection de types et de validité des variables

Nous pouvons utiliser l'opérateur « `typeof` » pour trouver le type d'une variable. Il nous retournera le type sous forme de string.

```
let prenom = "John";  
alert(typeof prenom); // Affiche string
```

```
let ok = true;  
alert(typeof ok); // Affiche boolean
```

Détection de types et de validité des variables

```
let prenom = "John";  
alert(typeof prenom == "string");  
// Affiche true
```

```
let ok = true;  
alert(typeof ok == "boolean");  
// Affiche true
```

Détection de types et de validité des variables

Que retourne chacun de ces exemples ?

1) `typeof "John"`

2) `typeof 3.14`

3) `typeof NaN`

4) `typeof false`

5) `typeof [1,2,3,4]`

6) `typeof {name:'John', age:34}`

7) `typeof new Date()`

8) `typeof function () {}`

9) `typeof null`

Détection de types et de validité des variables

Par contre, l'opérateur « **typeof** » ne pourra pas nous indiquer si notre objet est un tableau (array) ou une date par exemple.

Pour cela, nous pourrons utiliser la propriété « constructor ».

Détection de types et de validité des variables

<code>"John".constructor</code>	<i>// Returns "function String() { [native code] }"</i>
<code>(3.14).constructor</code>	<i>// Returns "function Number() { [native code] }"</i>
<code>false.constructor</code>	<i>// Returns "function Boolean() { [native code] }"</i>
<code>[1,2,3,4].constructor</code>	<i>// Returns "function Array() { [native code] }"</i>
<code>{name:'John', age:34}.constructor</code>	<i>// Returns "function Object() { [native code] }"</i>
<code>new Date().constructor</code>	<i>// Returns "function Date() { [native code] }"</i>
<code>function () {}.constructor</code>	<i>// Returns "function Function(){ [native code] }"</i>
<code>"John".constructor == String</code>	<i>// Returns true (ici String est l'objet String)</i>

TABLEAUX

Tableau

Ensemble de variables auxquelles on accède via un indice

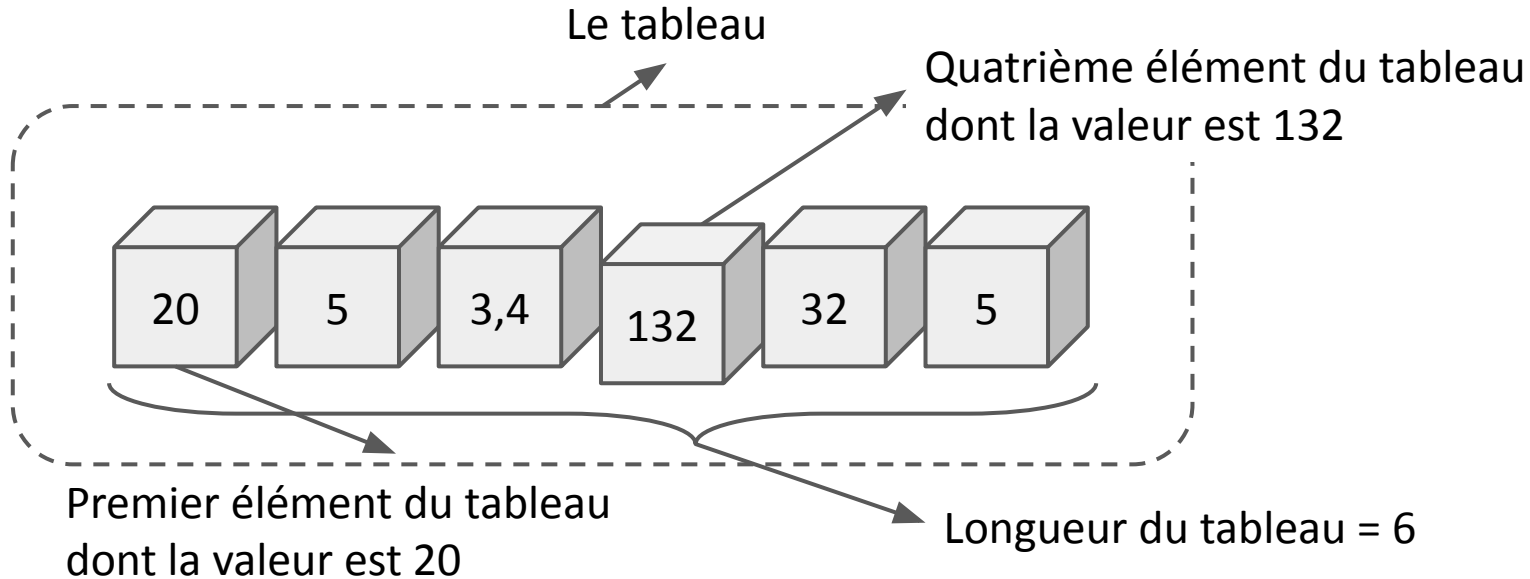
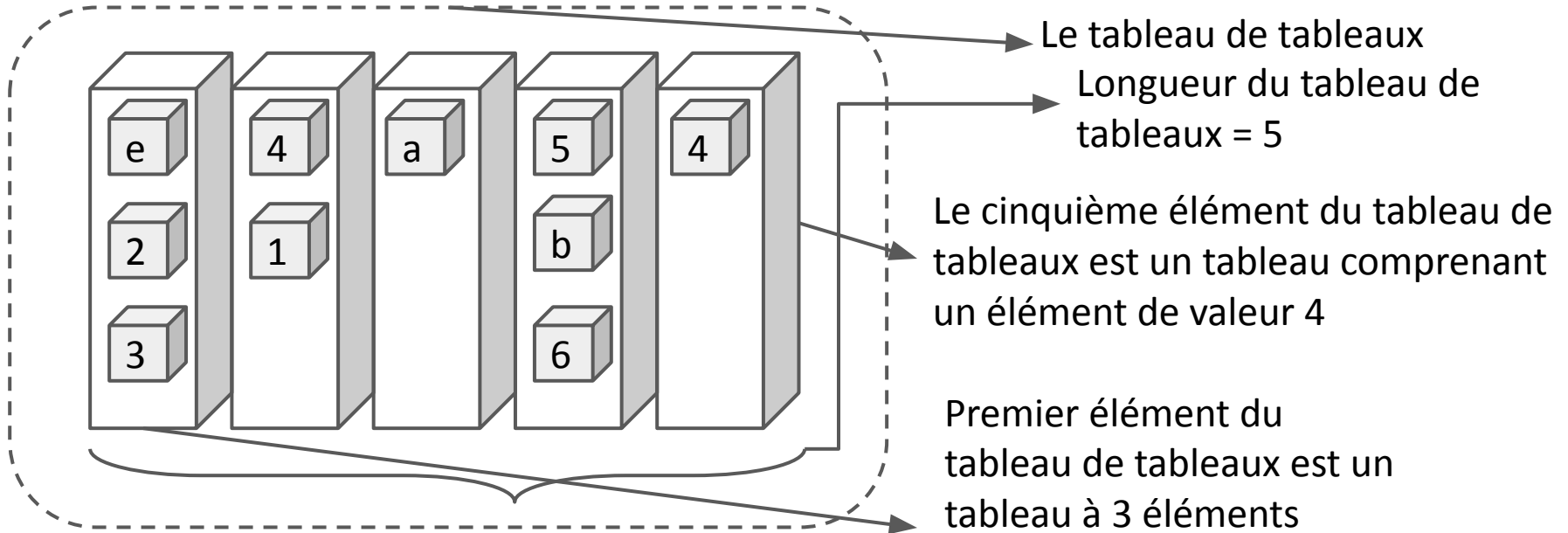


Tableau de tableaux

Ensemble de tableaux auxquels on accède via un indice



Tableaux en JavaScript

Tableaux multi-dimensionnels, associatifs

```
//Initialisation - Possibilité 1  
let monTableau = ["Pierre","Paul"];  
//Initialisation - Possibilité 2  
let monTableau2 = new Array("Pierre","Paul");  
//Index commence à 0  
monTableau[2] = "Jacques"; //ajoute un 3ième élément  
document.alert("Nom: " + monTableau[2]);
```

Tableaux en JavaScript

- Tableaux multi-dimensionnels, associatifs

```
//Tableau associatif : constructeur : Array[]  
let monTableau3 = new Array();  
monTableau3["Lundi"] = "Soupe";  
  
//Objet : constructeur : Object[]  
let monTableau4 = {"Mardi": "Pizza"};
```

Créer un tableau

- Les tableaux sont des objets, on peut donc les créer via le mot clé new

```
const tableau = new Array();
```

- On peut directement indiquer le nombre d'éléments constitutifs :

```
const tableau = new Array(42);
```

- Il est aussi possible de remplir le tableau lors de sa création

```
const tableau = new Array("toto", 42, mafonction());
```

Créer un tableau

- Le JavaScript nous permet également de créer un tableau « à la volée », sans le mot-clé new :

```
const tableau = ["Hello", 42, mafonction()];
```


Accéder à un élément d'un tableau

- Le JavaScript nous permet également d'accéder à un élément du tableau de deux manières différentes :

```
console.log(monTableau3["Lundi"]);
```

```
console.log(monTableau3.Lundi);
```

Modifier un élément d'un tableau

- Le JavaScript nous permet également de modifier un élément du tableau de deux manières différentes :

```
monTableau3["Lundi"] = "Pâtes";
```

```
monTableau3.Lundi = "Pâtes";
```

Fonctions propres aux tableaux

Il existe un certain nombre de fonctions utiles pour la manipulation des tableaux :

Fonction	
valueOf()	Renvoie la valeur primitive.
toString()	Renvoie les valeurs du tableau sous forme de string séparées par des virgules
join(*)	Idem que toString() mais on peut spécifier le séparateur
pop()	Retire le dernier élément du tableau
push()	Ajoute un élément à la fin du tableau

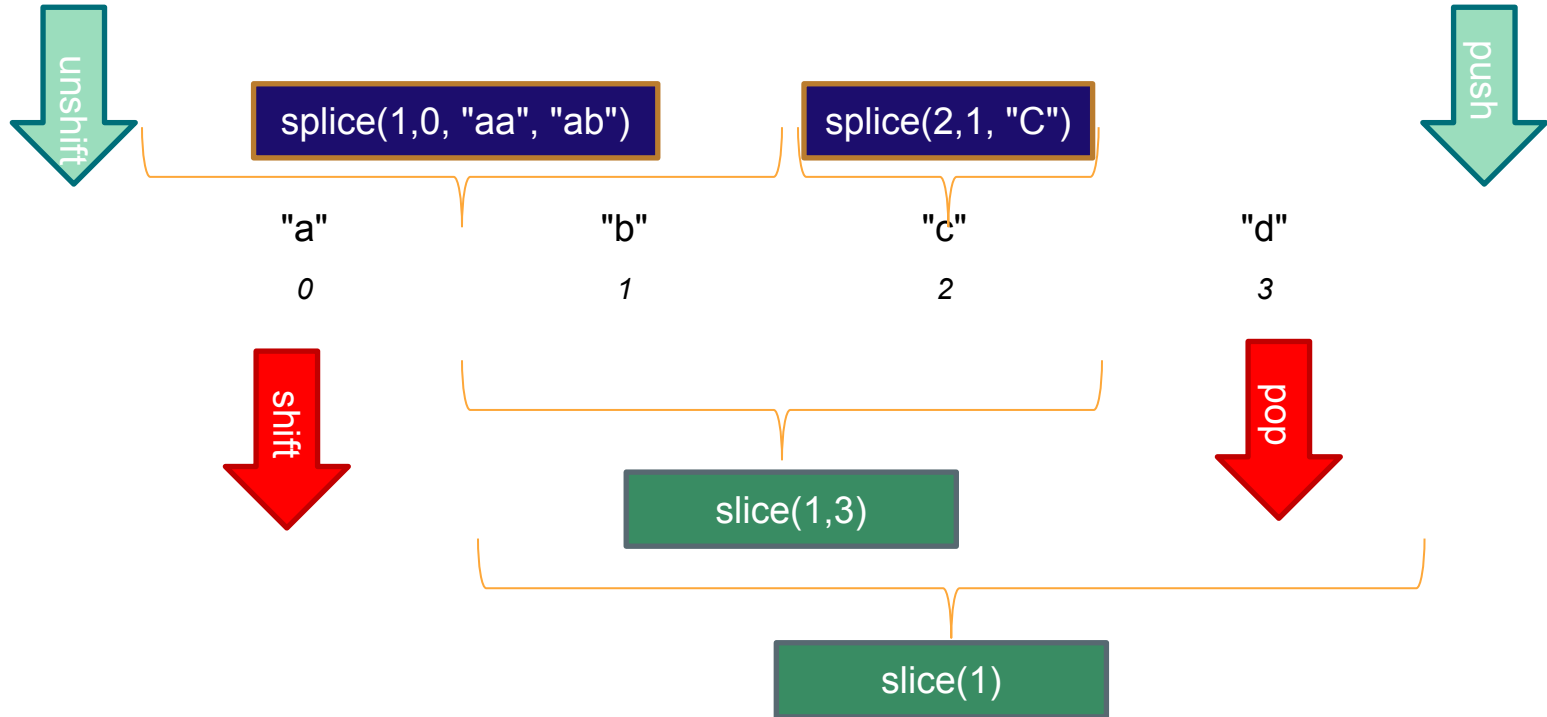
Fonctions propres aux tableaux

Fonction	
shift()	Retire le premier élément du tableau, et replace bien les éléments suivants.
unshift()	Ajoute un élément au début du tableau en réarrangeant les éléments suivants.
length	Retourne le nombre d'éléments du tableau
splice(a,b,el1,el2,...)	Permet d'ajouter les éléments 1, 2,... à partir de l'emplacement a, en supprimant b éléments.
sort()	Permet de trier alphabétiquement le tableau (attention, comportement différent avec des nombres)

Fonctions propres aux tableaux

Fonction	
<code>reverse()</code>	Inverse les éléments du tableau
<code>sort(function(a,b){return a-b})</code>	Permet de trier un tableau de nombres (croissant)
<code>sort(function(a,b){return b-a})</code>	Permet de trier un tableau de nombres (décroiss.)
<code>concat(tab2)</code>	Permet de concaténer le tab2 au tableau
<code>slice(a,b)</code>	Découpe le tableau entre l'élément indice a et l'élément indice b

Fonctions propres aux tableaux



CHAÎNES DE CARACTÈRES

Chaînes de caractères

Une chaîne de caractères se déclare simplement entre guillemets (simples ou doubles)

Exemple :

```
let car = "Volvo";
```

```
let car = 'Volvo';
```

```
let phrase = "Je m'appelle Jean";
```

```
let phrase = 'Il s\'appelle Jean';
```


Chaînes de caractères

Une chaîne de caractères peut également se déclarer comme un objet via le mot clé `new` :

```
let phrase = new String("Hello World");
```

Chaînes de caractères

Méthode	Paramètre	Description
<code>charAt</code>	Index du caractère dans la chaîne	Retourne le caractère localisé à l'index spécifié en paramètre.
<code>charCodeAt</code>	Index du caractère dans la chaîne	Retourne le code du caractère localisé à l'index spécifié en paramètre.
<code>concat</code>	Chaîne à concaténer	Concatène la chaîne en paramètres à la chaîne courante.
<code>fromCharCode</code>	Chaîne de caractères Unicode	Crée une chaîne de caractères en utilisant une séquence Unicode.
<code>indexOf</code>	Chaîne de caractères	Recherche la première occurrence de la chaîne passée en paramètre et retourne l'index de cette première occurrence.
<code>lastIndexOf</code>	Chaîne de caractères	Recherche la dernière occurrence de la chaîne passée en paramètre et retourne l'index de cette dernière occurrence.

Chaînes de caractères

Méthode	Paramètre	Description
<code>match</code>	Expression régulière	Détermine si la chaîne de caractères comporte une ou plusieurs correspondances avec l'expression régulière spécifiée.
<code>replace</code>	Expression régulière ou chaîne de caractères à remplacer puis chaîne de remplacement	Remplace un bloc de caractères par un autre dans une chaîne de caractères.
<code>search</code>	Expression régulière de recherche	Recherche l'indice de la première occurrence correspondant à l'expression régulière spécifiée.
<code>slice</code>	Index dans la chaîne de caractères	Retourne une sous-chaîne de caractères en commençant à l'index spécifié en paramètre et en finissant à la fin de la chaîne initiale si la méthode ne comporte qu'un seul paramètre. Dans le cas contraire, elle se termine à l'index spécifié par le second paramètre.
<code>split</code>	Délimiteur	Permet de découper une chaîne de caractères en sous-chaînes en se fondant sur un délimiteur.

Chaînes de caractères

Méthode	Paramètre	Description
<code>substr</code>	Index de début et de fin	Méthode identique à la méthode <code>slice</code>
<code>substring</code>	Index de début et de fin	Méthode identique à la précédente
<code>toLowerCase</code>	-	Convertit la chaîne de caractères en minuscules.
<code>toString</code>	-	Retourne la chaîne de caractère interne sous forme de chaînes de caractères.
<code>toUpperCase</code>	-	Convertit la chaîne de caractères en majuscules.
<code>valueOf</code>	-	Retourne la valeur primitive de l'objet. Est équivalente à la méthode <code>toString</code> .

Exercices

- Chaîne : « ma formation javascript »
- Avec la chaîne ci-dessus :
 - Retourner la position de « ma »
 - Indiquer l'indice de la lettre « p »
 - Retrouver la lettre située à l'indice 21
 - Remplacer « javascript » par « Java »
 - Découper la chaîne avec le délimiteur « » (espace)
 - Inverser la chaîne de caractères (+ difficile) :

« ma formation javascript » → « tpircsavaj noitamrof am »

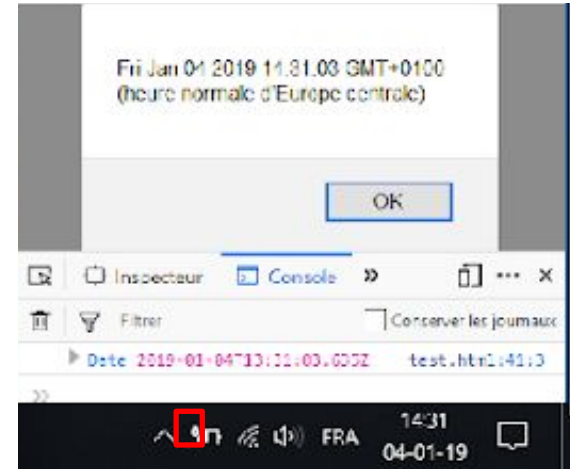
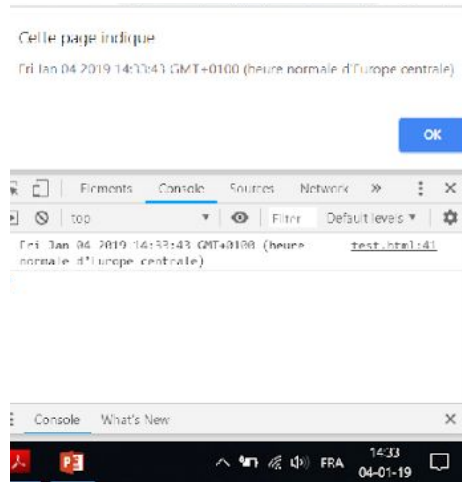
DATES

Date

- Le JavaScript nous propose un type d'objet particulier : les dates.

Exemple :

```
const today = new Date();  
console.log(today);  
alert(today);
```



La console de Firefox n'affiche pas en compte le fuseau horaire dans lequel la machine se trouve.

Date

Le constructeur Date(...) peut prendre plusieurs sortes d'arguments :

- aucun : date actuelle.
- nombre : temps 0 + n millisecondes
- 7 nombres : année, mois, jour, heure, minute, seconde et milliseconde

```
const d1 = new Date();           // 22/10/2019 12:40:14
const d2 = new Date(865734983959); // 08/06/1997 03:56:23
const d3 = new Date(2014,5,24,11,33,30,0); // 24/6/2014 11:33:30
const d4 = new Date(2013,11,25);    // 25/12/2013 0:00:00
```


Date – Méthodes utiles

Méthode	Description
<code>getFullYear()</code>	Donne l'année en 4 chiffres
<code>getMonth()</code>	Donne le mois (0-11)
<code>getDate()</code>	Donne le jour en tant que nombre (1-31)
<code>getDay()</code>	Donne le jour de la semaine (0-6) (<i>dimanche à samedi</i>)
<code>getHours()</code>	Donne l'heure (0-23)

Date – Méthodes utiles

Méthode	Description
getMinutes()	Donne les minutes (0-59)
getSeconds()	Donne les secondes (0-59)
getMilliseconds()	Donne les millisecondes (0-999)
getTime()	Donne le temps absolu (millisecondes depuis le 1/1/1970)

LES OPÉRATEURS

Opérateurs

Opérateurs de calcul

+	// Addition
-	// Soustraction
*	// Multiplication
/	// Division
%	// Modulo

Opérateurs

Opérateurs d'affectation

```
=      // Simple affectation
+=     // Ajoute ce qu'il y a à droite
-=     // Retire ce qu'il y a à droite
*=     // Multiplie par ce qu'il y a à droite
/=     // Divise par ce qu'il y a à droite
%=     // Modulo de la division entière par ce qu'il y a à droite
```

Opérateurs

- Opérateurs de comparaison

`>, <, >=, <=`

- La comparaison de chaînes se fonde sur les codes des caractères (note : `a > A`)
- La comparaison de chaîne et nombre convertit automatiquement la chaîne en nombre

Si la conversion échoue, la comparaison s'effectue avec NaN et renvoie false

Opérateurs

- Opérateurs d'égalité

`==`

`===` `// Égalité de type et de valeur`

`!=`

`!==`

- Opérateur de concaténation

`+`

```
alert("Ma "+"chaîne "+"concaténée.");
```

Opérateurs

- Opérateurs logiques

`!` `// Négation`

`&&` `// ET logique`

`||` `// OU logique`

- Opérateur conditionnel

Permet d'initialiser une variable dont la valeur se fonde sur le résultat d'une condition.

```
let maVariable = (cond)?(si_true):(si_false);
```


Opérateurs

- Opérateurs unaires

typeof

Détermine le type d'une variable sous forme d'un string

new

Création d'un objet

delete

Permet de retirer une propriété donnée d'un objet

Opérateurs

Exemple Delete:

```
const Employee = {  
    age: 28,  
    nom: "John",  
    designation: "developpeur "  
}  
  
console.log(Employee.nom);    //renvoie "John"  
  
delete Employee.nom  
  
console.log(Employee.nom);    //renvoie undefined
```

Opérateurs

- Void

Souvent utilisé pour obtenir la valeur undefined avec « void(0) ».

Dans le cadre d'un URI qui est évalué, le résultat remplace le contenu de la page, sauf si la valeur renvoyée vaut undefined.

```
<a href = "javascript:void(0);"> cliquer (sans effet) </a>
```

```
<a href="javascript:void(document.body.style.backgroundColor='green');">
```

```
Cliquer ici pour rendre le fond vert </a>
```

Opérateurs

- Post-Incrémentation (variable++)

```
let x = 3;
```

```
y = x++;
```

Résultat : $y = 3$, $x = 4$

- Prè-Incrementation (++variable)

```
let x = 3;
```

```
y = ++x;
```

Résultat : $y = 4$, $x = 4$

Exercice

- Calcul de la TVA

Écrire un programme qui :

1. Demande à l'utilisateur un prix unitaire hors taxe d'un livre
2. Demande à l'utilisateur la quantité de livre
3. Calcule et affiche le prix total TTC de la commande, en utilisant une TVA de 21%

Pour interagir avec l'utilisateur, vous utiliserez les fonctions d'entrée/sortie `prompt()` et `alert()`.

STRUCTURE DE CONTRÔLE ET EXCEPTIONS

if...else...

Conditions

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

switch

Switch case

```
switch( variable ) {  
  case valeur:  
    ...  
    break;  
  default:  
    ...  
}
```


switch

```
switch (expr) {  
  case "Oranges":  
    console.log("Oranges : 0.59 € le kilo.");  
    break;  
  case "Pommes":  
    console.log("Pommes : 0.32 € le kilo.");  
    break;  
  case "Bananes":  
    console.log("Bananes : 0.48 € le kilo.");  
    break;  
  case "Cerises":  
    console.log("Cerises : 3.00 € le kilo.");  
    break;  
  case "Mangues":  
  case "Papayes":  
    console.log("Mangues et papayes : 2.79 € le kilo.");  
    break;  
  default:  
    console.log("Désolé, nous n'avons plus de " + expr + ".");  
}
```

Boucle for

Boucle for

```
for( let cpt=0; cpt<10; cpt++ )  
{  
    ...  
}
```

The diagram illustrates the three components of a JavaScript for loop. Three light blue boxes at the bottom are connected by lines to the for loop syntax above. The first box, 'Traitements d'initialisation', has an arrow pointing to 'let cpt=0;'. The second box, 'Condition de séjour', has an arrow pointing to 'cpt<10;'. The third box, 'Traitements à effectuer après chaque itération', has an arrow pointing to 'cpt++'.

Traitements d'initialisation

Condition de séjour

Traitements à effectuer après
chaque itération

Boucle for

Boucle for

```
for( variable in structure ) {  
    (...)  
}
```

```
let objet = {a: 1, b: 2, c: 3};  
for( let propriete in objet ) {  
    alert(propriete + ": " + objet[propriete]);  
}  
  
//3 fois alert [a:1], [b:2], [c:3]
```

Boucle while

Boucle while

```
while(condition de séjour) {  
    (...)  
}
```

```
let nombre = 0;  
while( nombre < 10 ) {  
    ...  
    nombre++;  
}
```

Boucle do .. while

Boucle do ... while

```
do {  
    (...)  
} while( condition de séjour );
```

```
let nombre = 0;  
do {  
    ...  
    nombre++;  
} while( nombre < 10 );
```

break et continue

Le mot clé **break** permet de sortir d'une boucle.

```
for (let i = 0; i < 10; i++) {  
  if (i === 3) { break; }  
  text += "The number is " + i + "<br>";  
}
```

The number is 0
The number is 1
The number is 2

Le mot clé **continue** permet de sortir d'une itération.

```
for (let i = 0; i < 10; i++) {  
  if (i === 3) { continue; }  
  text += "The number is " + i + "<br>";  
}
```

The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

Exercice

- Utilisez l'objet Date et des structures conditionnelles, écrivez un programme qui affiche le jour de la semaine.

Exemple : « Bonjour, nous sommes lundi! »

Exercice

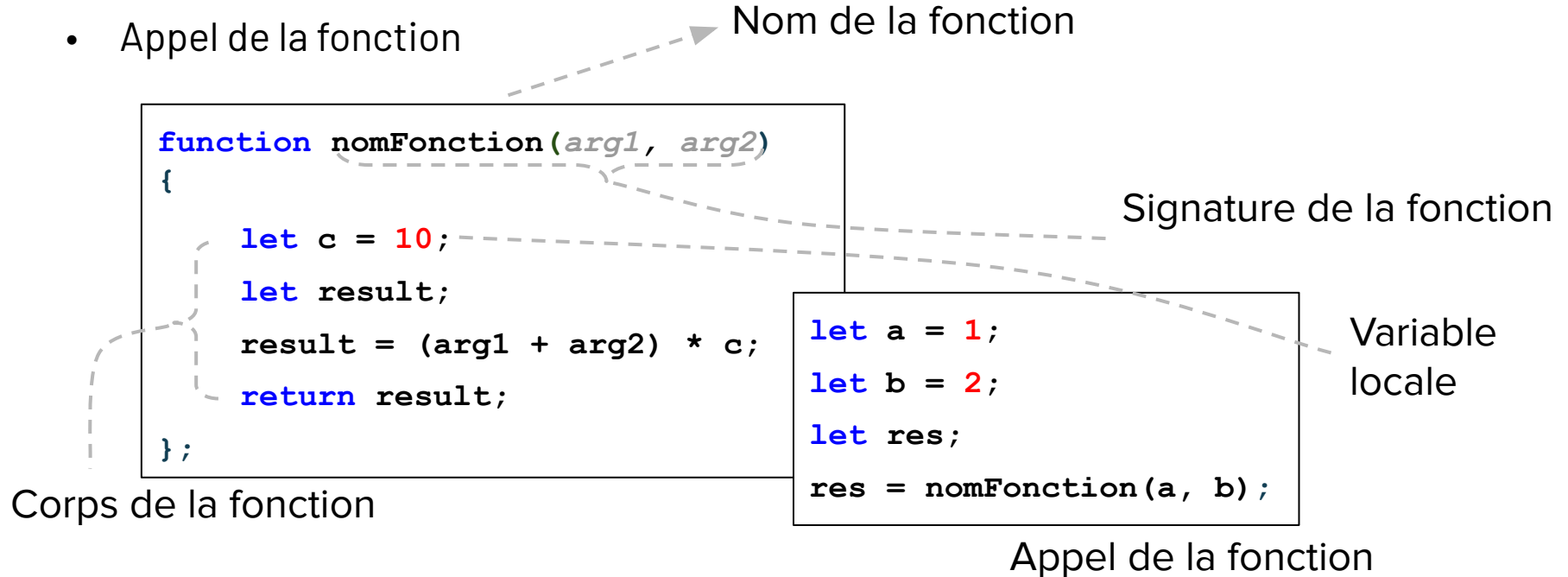
- Réalisez un programme qui permet d'afficher, dans la console, la structure suivante à l'aide d'une boucle :

```
"A "  
"AA "  
"AAA "  
"AAAA "  
"AAAAA "  
"AAAAAA "  
"AAAAAAA "  
"AAAAAAA "  
"AAAAAAA "  
"AAAAAAA "
```


FONCTIONS

Structure des applications - Fonctions

- Déclaration de la fonction
- Appel de la fonction



Structure des applications - Fonctions

- Paramètres formels: Paramètres utilisés dans le corps de la fonction
 - o Ex: arg1, arg2 sur le slide précédent
- Paramètres effectifs: Variables utilisées lors de l'appel d'une fonction
 - o Ex: a, b sur le slide précédent

Votre première fonction !

Créez une fonction *inverser(chaine)* qui effectuera une inversion des caractères d'une chaîne et affichera le résultat en console et en alerte.

Pour rappel : Inverser les caractères se fait en 3 étapes !

Structure des applications - Fonctions

```
function maFonction(a, b, c, d, e, f){  
    let res;  
    res = a + b;  
    res = res * d;  
    e = f;  
    res = res - e;  
    return res;}  
  
let a = 2;  
let b = 3;  
let c = 4;  
let d = 5;  
let e = 6;  
let f = 7;  
let g = maFonction(f, e, d, c, b, a);
```

Que valent g et e après l'appel
de la fonction?

Structure des applications - Fonctions

Arguments:

- Type simple et string passés par valeur
- Type complexe passé par référence
- Tous les arguments sont optionnels

Valeur vs référence

```
<script type="text/javascript">  
function emptyMe(arg1)  
{  
  arg1.value = "";  
}  
</script>
```

 **Objet**

OK!

```
...  
<input type="text" value="Howdy" onchange="emptyMe(this)">
```

```
<script type="text/javascript">  
function emptyMe(arg1)  
{  
  arg1 = "";  
}  
</script>
```

 **Valeur**

KO!

```
...  
<input type="text" value="Howdy" onchange="emptyMe(this.value)">
```

Fonctions anonymes

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

Permet de créer des « variables » contenant des fonctions.

```
let (x) = function (arguments) {  
  // Le code de votre fonction anonyme  
};  
x(arguments);
```


Fonctions anonymes

Les fonctions anonymes ont également une utilité pour isoler son code :

```
(function () {  
  // Code isolé ...  
}) ();
```

Fonctions fléchées

La fonction fléchée est un raccourci de syntaxe :

// ES 6

```
let fonctionPlusUn = x => x + 1;
```



// ES5

```
let fonctionPlusUn = function(x) {  
    return x + 1;  
}
```

paramètre

retour

Gestion des exceptions

Il est possible de « tester » du code en utilisant la structure try/catch.

Le navigateur va essayer d'exécuter le code situé dans le bloc try. Le bloc catch contient le comportement à avoir lorsqu'une erreur est lancée.

Gestion des exceptions

Le mot clé `throw` permet de gérer ses propres messages d'erreurs personnalisés.

`Throw` lance une exception. Cette exception est soit une chaîne de caractères, soit un chiffre ou encore un booléen.

Gestion des exceptions

```
<p>Please input a number between 5 and 10 : </p>

<input id='demo' type='text' />
<button type='button' onclick="myFonction()">Test Input</button>
<p id='message'></p>

<script type="text/javascript">
  function myFonction() {
    var message, x;
    message = document.getElementById('message');
    message.innerHTML = "";
    x = document.getElementById('demo').value;
    try {
      if(x == "") throw 'Empty';
      if(isNaN(x)) throw 'Not a number';
      x = Number(x);
      if(x < 5) throw 'Too low';
      if(x > 10) throw 'Too high';
    }
    catch(err) {
      message.innerHTML = 'Input is : ' + err;
    }
  }
</script>
```

L'API du DOM

Document Object Model

DOM est une norme pour que tous les navigateurs affichent la même chose.

Avant cette norme, chacun avait sa propre mise en forme. Le W3C a normalisé ça

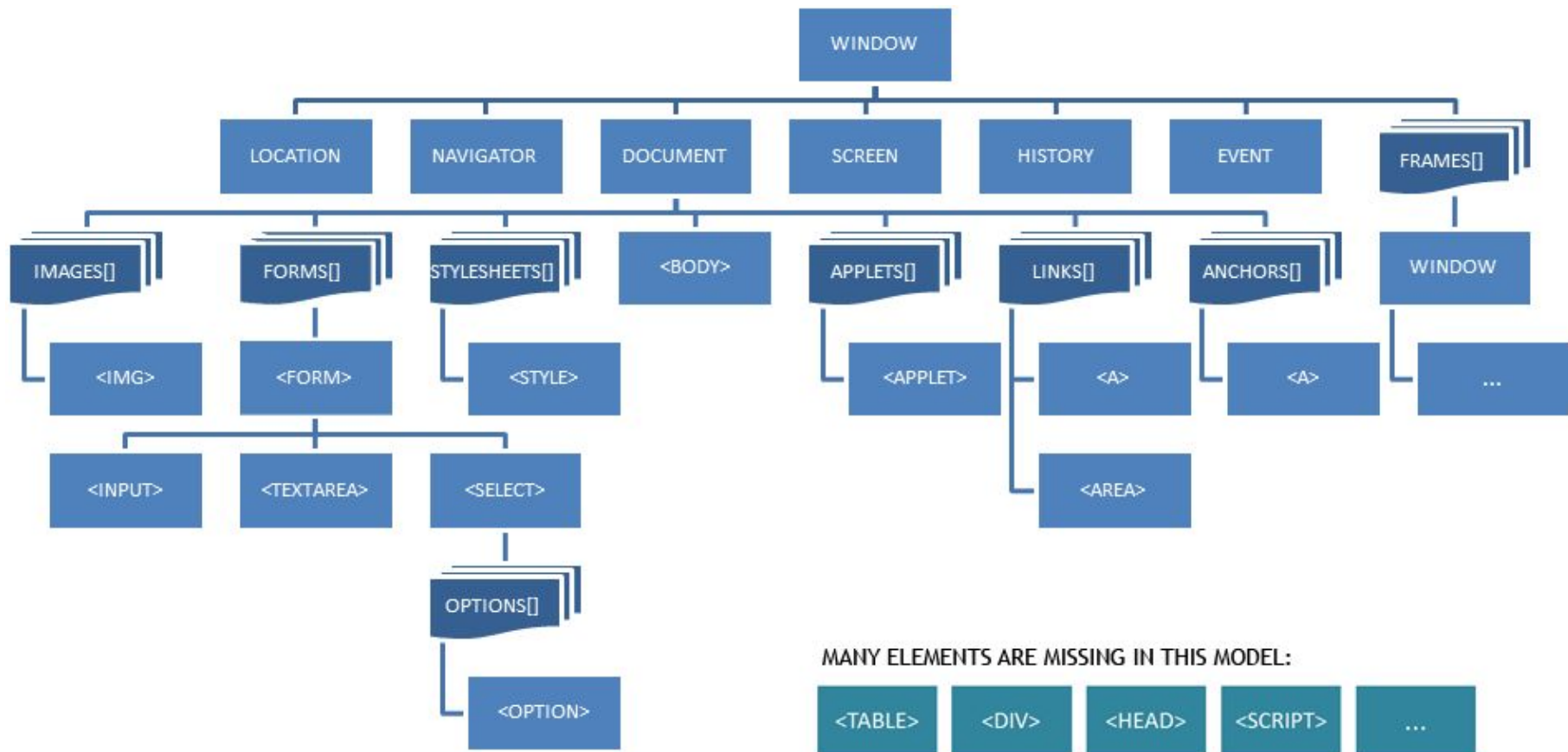
	1998	2000	2004	2014
DOM 0	DOM 1	DOM 2	DOM 3	DOM 4
Mis à disposition par Netscape Navigator 2.0	Précision d'un document XML sous forme d'arbre composé de nœud Fonction pour se déplacer dans l'arbre Gestion des formulaires Dès IE5 et Netscape 6 Core	Core HTML Events Style View Transversal & Range getElementById	XPath Événements claviers Sérialisation XML	En cours de développement

Document Object Model

L'API du DOM (Document Object Model) permet d'accéder à une page Web et de manipuler son contenu, sa structure ainsi que ses styles

DOM présente un document sous la forme d'un arbre de noeuds

https://developer.mozilla.org/en-US/docs/DOM/DOM_Reference



Objets du DOM

- L'arbre se compose d'un ensemble de nœuds:

- les nœuds élément ou `NODE.ELEMENT_NODE`
- les nœuds texte ou `NODE.TEXT_NODE`

```
<html>
<head>
<title>Mon titre</title>
</head>
<body>
  <p> Mon texte </p>
</body>
</html>
```

Attention, qu'un retour à la ligne est considéré comme un nœud texte !!!

Objets du DOM

window : Représente la fenêtre du navigateur où le code HTML est affiché.

navigator : Objet en lecture seule. Représente les caractéristiques du navigateur utilisé.

screen : Objet en lecture seule permettant d'accéder aux caractéristiques de l'écran affichant la page.

location : Cet objet possède les caractéristiques de la page actuelle (url, protocole, ...)

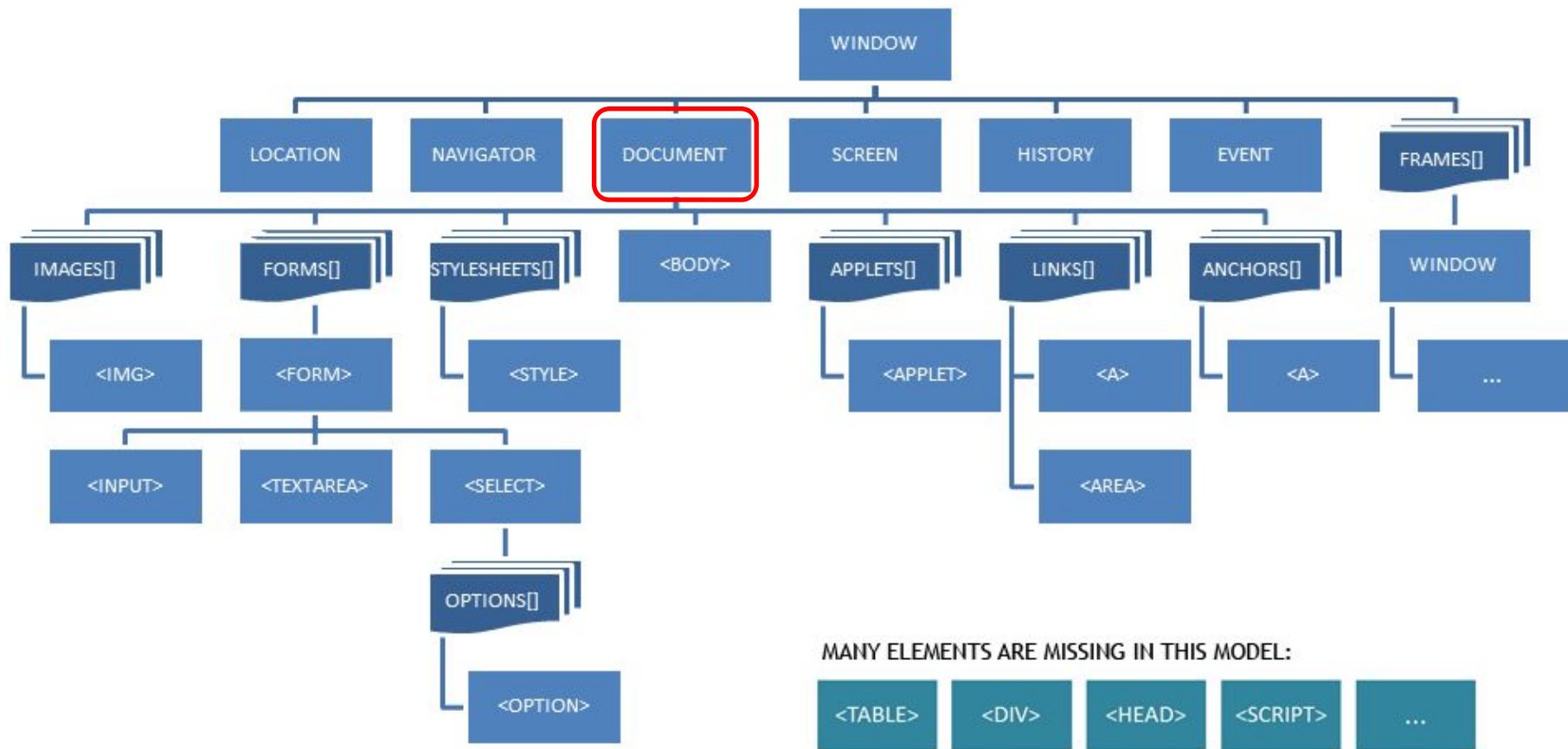
document : C'est l'objet qui contient toute notre page HTML <html></html>. Cet objet possède une très grande quantité de propriétés et fonctions.

Afficher les propriétés d'un objet

```
function afficherProprietes (obj) {  
  for(let prop in obj) {  
    let valeur = obj[prop];  
    document.getElementById("info").innerHTML +=  
      "<br>" + prop + ":" + valeur;  
  }  
}  
afficherProprietes (window) ;
```

Nommer les objets

- Pour manipuler les différents objets d'une page Web (paragraphes, divs, images, ...), il faut pouvoir les différencier.
- On utilise très souvent l'id pour nommer un élément.
- Pour rappel : l'id d'un élément doit être unique au sein de la page.



DOM Document

DOM Document

L'objet Document est l'élément racine d'un document (page Web, document XML, ...)

Il hérite des méthodes et propriétés de l'objet Noeud

Ajouter au DOM

L'API du DOM

DOM Document - Méthodes

write(*string*)

Écrit dans la page. À n'utiliser que lors du chargement de la page !
Écrase tout ce qui se trouve déjà dans la page.

Paramètres:

string: STRING

Return:

/

Exemple:

```
document.write("Hello World");
```

Récupérer un élément du DOM

L'API du DOM

DOM Document - Méthodes

getElementById(*elementID*)

Retourne l'élément ayant l'attribut id spécifié

Paramètres:

elementID: STRING

Return:

ELEMENT (OBJET !!)

Exemple:

```
const maDiv = document.getElementById("demo");
```

Récupérer une collection d'éléments

L'API du DOM

DOM Document - Méthodes

getElementsByClassName(*class*)

Retourne une **collection** d'éléments ayant la classe spécifiée

Paramètres:

elementID: STRING

Return:

ELEMENT (OBJET !!)

Exemple:

```
const maDiv = document.getElementsByClassName ("demo") ;
```

DOM Document - Méthodes

getElementsByTagName(*tagName*)

Retourne une **collection** d'éléments ayant le nom spécifié

Paramètres:

tagName: STRING

Return:

NODELIST (OBJET !)

Exemple:

```
document.getElementsByTagName ("h1") ;
```

Créer un élément DOM

L'API du DOM

DOM Document - Méthodes

createElement(*nodeName*)

Retourne l'élément créé. Ne le place pas dans la page.

Paramètres:

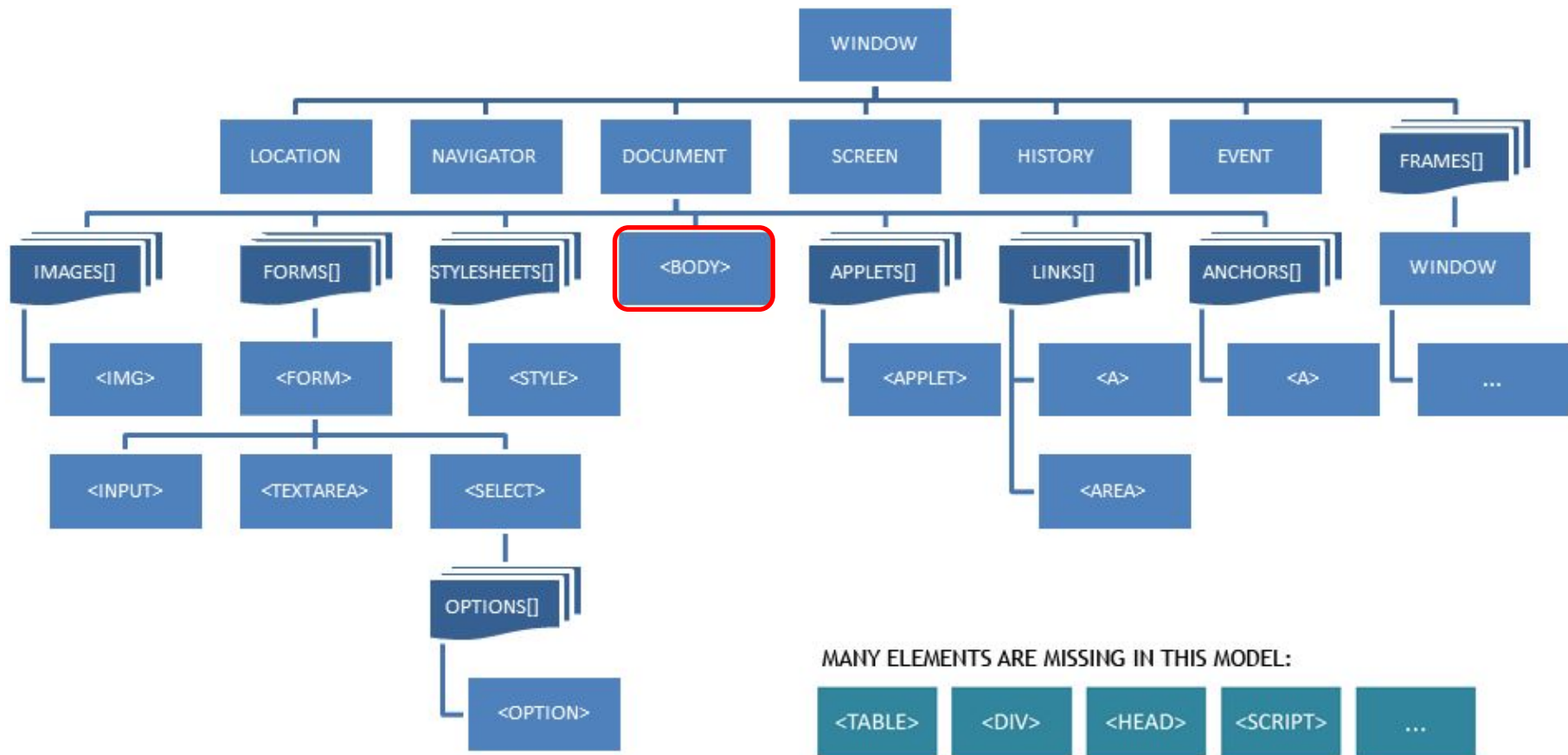
nodeName: STRING

Return:

ELEMENT

Exemple:

```
const titre = document.createElement("h1");  
document.body.appendChild(titre);
```

DOM Node

DOM Node

Représente un noeud dans un document HTML

Il existe 12 types de noeuds HTML, dont:

- Element : `Node.nodeType = 1`
- Attr : `Node.nodeType = 2`
- Text : `Node.nodeType = 3`
- Comment : `Node.NodeType = 8`

<https://developer.mozilla.org/en-US/docs/Web/API/Node.nodeType>

Récupérer des informations

L'API du DOM

DOM Node - Propriétés

attributes

Retourne une **collection** (NamedNodeMap) contenant les attributs d'un nœud

Ex : `let x = document.getElementById("myBtn").attributes.length;`

childNodes

Retourne une **collection** (NodeList) contenant les noeuds enfants d'un nœud

Ex : `const c = document.body.childNodes;`

firstChild

Retourne le premier **noeud** enfant d'un nœud

Ex : `let x = document.getElementById("myList").firstChild.innerHTML;`

DOM Node - Propriétés

lastChild

Retourne le dernier **noeud** enfant d'un noeud

Ex : `let x = document.getElementById("myList").lastChild.innerHTML;`

parentNode

Retourne le **noeud** parent

Ex : `let x = document.getElementById("myLI").parentNode.nodeName;`

nodeName

Retourne le **nom** d'un noeud, c-à-d.:

- Le nom de la balise pour les noeuds Element
- Le nom de l'attribut pour les attributs
- ...

DOM Node - Propriétés

previousSibling

Retourne le **noeud** précédent du niveau identique au noeud courant

Ex : `let x = document.getElementById("item2").previousSibling.innerHTML;`

nextSibling

Retourne le **noeud** suivant du niveau identique du noeud courant

Ex : `let x = document.getElementById("item2").nextSibling.innerHTML;`

textContent

Retourne le **texte** d'un noeud et de ses descendants

Ex : `let x = document.getElementsByTagName("BUTTON")[0].textContent;`

DOM Node - Propriétés

nodeType

Retourne le **type** d'un nœud, c-à-d.:

- 1 pour les nœuds Element
- 2 pour les attributs
- 3 pour le texte
- ...

Ex : `let x = document.getElementById("myP").nodeType;`

nodeValue

Retourne la **valeur** d'un nœud (/!\ null si `nodeType == 1`)

Ex : `let x =`

`document.getElementsByTagName("BUTTON")[0].childNodes[0].nodeValue;`

Ajouter un nœud

L'API du DOM

DOM Node - Méthodes

appendChild(*node*)

Ajoute un nœud en tant que dernier enfant

Paramètres:

node: NODE

Return:

NODE

Exemple:

```
let node=document.getElementById("myList2").lastChild;  
document.getElementById("myList1").appendChild(node);
```

DOM Node - Méthodes

insertBefore(*newNode*, *existingNode*)

Ajoute un **noeud** juste avant le noeud enfant spécifié

Paramètres:

newNode: NODE

existingNode: NODE

Return:

NODE

Ex : list.**insertBefore**(newNoeud, noeudExistant);

DOM Node - Méthodes

insertAdjacentHTML(*position*, *texte*)

Ajoute un **noeud** selon la position spécifiée

Paramètres:

position: TEXT

texte: TEXT

<!-- **beforebegin** --> : Avant l'élément lui-même

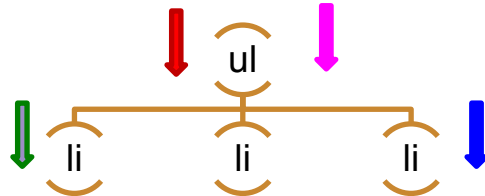
<!-- **afterbegin** --> : Juste à l'intérieur de l'élément, avant son premier enfant

Elt 1Elt 2Elt 3

<!-- **beforeend** --> : Juste à l'intérieur de l'élément, après son dernier enfant

<!-- **afterend** --> : Après l'élément lui-même

Ex : `ul.insertAdjacentHTML('beforebegin', '<h1>Titre</h1>');`



Remplacer un nœud

L'API du DOM

DOM Node - Méthodes

replaceChild(*node*)

Remplacer un **nœud** par un autre spécifié

Paramètres:

node: NODE

Return:

NODE

Exemple:

```
let li = document.createElement("li");  
li.textContent = "Bonjour";  
let ul = document.getElementById("myList");  
let li1 = ul.childNodes[1];  
ul.replaceChild(li, li1);
```

```
<ul id="myList">  
  <li>1</li>  
  <li>2</li>  
</ul>
```

Ne pas oublier de compter
le retour à la ligne

Supprimer un nœud

L'API du DOM

DOM Node - Méthodes

removeChild(*node*)

Supprime le **nœud** enfant spécifié

Paramètres:

node: NODE

Return:

NODE

Exemple:

```
const list=document.getElementById("myList");  
list.removeChild(list.childNodes[0]);
```

DOM Élément

DOM Element

Manipuler les attributs

Méthode	Paramètre	Description
getAttribute	Identifiant de l'attribut	Référence un attribut d'un élément en utilisant son identifiant.
hasAttribute	Identifiant de l'attribut	Détermine si un attribut est présent pour un élément.
removeAttribute	Identifiant de l'attribut	Supprime un attribut pour un élément.
setAttribute	Identifiant de l'attribut ainsi que sa valeur	Crée un attribut ou remplace un attribut existant d'un élément.

DOM Element - Propriétés

previousElementSibling

Retourne le **noeud** élément précédent du niveau identique du noeud courant

Ex : let x =

```
document.getElementById("item2").previousElementSibling.innerHTML;
```

nextElementSibling

Retourne le **noeud** élément suivant du niveau identique du noeud courant

Ex : let x = document.getElementById("item2").nextElementSibling.innerHTML;

DOM HTML Element

innerHTML

Cet attribut permet **d'accéder** ou de **remplacer** complètement le contenu d'un élément par celui spécifié dans une chaîne de caractères.

```
<div id="demo">  
  <p>P1</p>  
</div>
```

```
<div id="demo">  
  Paragraph  
  changed  
</div>
```

Ex : `document.getElementById("demo").innerHTML = "Paragraph changed!";`

Ex : `let x = document.getElementById("myP").innerHTML;`

`console.log(x);`

```
<p>P1</p>
```

DOM HTML Element

outerHTML

Cet attribut permet **d'accéder** ou de **remplacer** complètement le contenu d'un élément par celui spécifié dans une chaîne de caractères avec la balise parente.

```
<div id="demo">  
  <p>P1</p>  
</div>
```

Paragraph changed

Ex : `document.getElementById("demo").outerHTML = "Paragraph changed!"`;

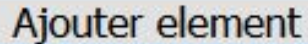
Ex : `let x = document.getElementById("myP").outerHTML`;

`console.log(x);` `<div id="demo"><p>P1</p></div>`

Exercice

1. Créez un programme qui permet d'ajouter un élément à une liste lorsque l'on clique sur un bouton.

- item



Ajouter element

Rappels utiles :

```
document.createElement(TypeElement); // Créer un élément
element.innerHTML = "chaîne";         // Modifier le contenu
element.appendChild(element);         // Ajouter à la suite
```

Exercice

2. En reprenant le code précédent, permettre à l'utilisateur d'écrire l'élément à l'aide d'un input.

- test

Faites attention à ce que le champ soit rempli ;-)

Exercice

3. Créez le programme permettant d'ajouter des articles dans un panier. Il faut pouvoir calculer le total des achats en temps réel.

Descriptif Produit

Chaise	25	Ajouter
Table	150	Ajouter
Meuble TV	250	Ajouter

Mon panier

Table	150
Table	150
Meuble TV	250
Meuble TV	250
Chaise	25

Prix Total : 825

Exercice

4. Reprenez le programme du panier.
Ajoutez la possibilité de supprimer un article du panier.
Il faudra bien sûr recalculer le total.

Descriptif Produit

Chaise	25	Ajouter au panier	Retirer du panier
Table	150	Ajouter au panier	Retirer du panier
Meuble TV	250	Ajouter au panier	Retirer du panier

Mon panier

Chaise	25	1
Table	150	0

Prix Total : 25

Via le CSS

DOM Élément

DOM HTML Element

querySelector(cssSelector)

Retourne le **premier élément** satisfaisant le selecteur CSS

Paramètres:

CSSselector : STRING

Return:

ELEMENT

Exemple:

```
document.querySelector(".example").style.backgroundColor = "red";
```

DOM HTML Element

querySelectorAll(cssSelector)

Retourne une **collection** d'éléments satisfaisants le sélecteur CSS

Paramètres:

CSSselector : STRING

Return:

NODELIST

Exemple:

```
let x = document.querySelectorAll(".example");  
x[0].style.backgroundColor = "red";
```

Via les classes

DOM Élément

DOM HTML Element

classList

Renvoie la liste des classes que contient un élément.

Return:

Object : DOMTokenList []

```
<h1 class="avant">Titre</h1>
```

Exemple:

```
let h1 = document.querySelector("h1");  
console.log(h1.classList);
```

DOM HTML Element

classList.add(classe) / classList.remove(classe)

Ajoute/supprime une classe à un élément.

Paramètres:

classe : STRING

```
<h1 class="avant">Titre</h1>
```

Return:

/

Exemple:

```
let h1 = document.querySelector("h1");  
h1.classList.remove("avant");  
h1.classList.add("apres");
```

DOM HTML Element

classList.contains(classe)

Revoie True si l'élément a la classe.

Paramètres:

classe : STRING

Return:

Bool

Exemple:

```
let h1 = document.querySelector("h1");  
console.log(h1.classList.contains("avant"));  
console.log(h1.classList.contains("apres"));
```

```
<h1 class="avant">Titre</h1>
```

Les Timers

Les différents Timers

Les timers nous permettent d'exécuter des actions après un certain délai ou d'exécuter des actions toutes les n secondes.

```
setTimeout( fonction, compte à rebours)
```

```
setInterval(fonction, compte à rebours)
```

Les timers - Timeout

setTimeout

Prend deux paramètres:

- la fonction à exécuter
- le délai en millisecondes

Le délai est un compte à rebours qui permettra de déclencher la fonction.

Une fois l'exécution de la fonction lancée, le compte à rebours n'est pas déclenché à nouveau.

Ex : `setTimeout(function(){ alert("Hello"); }, 3000);`

Les timers - Timeout

clearTimeout

Prend en paramètre le timer (généralement stocké dans une variable).

Permet l'arrêt du compte à rebours avant le déclenchement de la méthode.

```
Ex : function myStopFunction() {  
    clearTimeout(myVar);  
}
```

Les timers - Interval

setInterval

Prend deux paramètres:

- la fonction à exécuter
- le délai en millisecondes

Le délai est un compte à rebours qui permettra de déclencher la fonction.

Ici la différence c'est que le compte à rebours se déclenchera en boucle.

Ex : `setInterval(function(){ alert("Hello"); }, 3000);`

Les timers - Interval

clearInterval

Prend en paramètre le timer (généralement stocké dans une variable).

Permet l'arrêt du compte à rebours.

Ex: `clearInterval(maVarTimer);`

Exercice

1. Affichez l'heure actuelle (heure : minutes : secondes) dans le titre de la fenêtre de votre navigateur en utilisant `setTimeout` et ensuite `setInterval`.
2. Affichez la date et l'heure sur votre page web.

Mardi 25 Avril

13:16:32

MANIPULER LE CSS

Manipuler le CSS

S'il est possible de manipuler le code HTML, il est également possible de modifier le CSS d'une page web !

```
document.getElementById("toto").style.backgroundColor = "red";
```


Manipuler le CSS

La manipulation du CSS se fait à l'aide de l'attribut style qui est propre aux éléments.

L'attribut style possède toutes les propriétés CSS, il faut néanmoins faire attention aux changements de nom dans certains cas.

```
element.style
```

```
.background = "#f3f3f3 url('img.png') no-repeat right top";  
.backgroundColor = "red";  
.display = "none";  
.fontSize = "35px";  
.position = "absolute";  
.top = "210px";  
.left = "100px";
```

Manipuler le CSS

Le seul « problème », c'est que le style CSS est ajouté dans le code HTML. Il s'agit de style in-line !

Essayez de modifier le CSS d'un élément, remarquez que le style s'applique avec l'attribut `style="..."`

Limites

Essayez le code suivant (sur un élément stylisé) :

```
const elem = document.getElementById("toto");  
elem.style.color = "red";  
console.log(elem.style.color);
```

Quel est le message affiché en console ?

Limites

Le JavaScript ne sait lire que les propriétés CSS inline !

Il faudra utiliser la fonction `getComputedStyle()` pour lire le CSS externe.

Fonctionnement :

```
const elem = document.getElementById("toto");  
let styles = getComputedStyle(elem);  
let color = styles.color;
```

Pour faire plus rapide :

```
let color = getComputedStyle(document.getElementById("toto")).color;
```

Limites

Pour récupérer le positionnement, il faut s'y prendre autrement. On utilise les propriétés offset.

- `offsetWidth` `// width+padding+border !`
- `offsetHeight` `// height+padding+border !`
- `offsetLeft`
- `offsetTop`
- `offsetParent` `/* contient l'objet de l'élément parent
par rapport auquel est positionné l'élément actuel. */`

Les Évènements

Événements

- Les objets du DOM peuvent réagir à des événements. L'exemple le plus commun d'événement est le clic sur un élément.

Exemple :

```
<button onclick="afficherText()">Cliquer ici</button>  
<p id="info"></p>
```

```
function afficherText() {  
    document.getElementById("info").innerHTML = "Hello World";  
}
```

Gérer les événements

- Ajouter une action à un élément pour un événement :

```
document.getElementById("toto").onclick = maFonction;
```

OU

```
document.getElementById("toto").addEventListener("click",maFonction);
```

Notez l'absence de parenthèses !

Gérer les événements

- Supprimer une action à un élément pour un événement :

```
document.getElementById("toto").removeEventListener("click",maFonction);
```

Notez l'absence de parenthèses !

Gérer les événements

Si l'on veut passer des arguments à la fonction qui est lancée, il faut passer par une fonction intermédiaire :

```
document.getElementById("toto").onclick = function (e){  
    maFonction(arg1, arg2);  
}
```

Type d'événements

Descriptif	addEventListener(...)	Ajout direct
Le contenu d'un champ change	change	onchange
Clic de souris sur un élément	click	onclick
Double clic sur un élément	dblclick	ondblclick
Si une erreur apparaît lors du chargement de la page, d'une image...	error	onerror
L'élément reçoit le focus	focus	onfocus
L'élément perd le focus	blur	onblur

Type d'événements

Descriptif	<code>addEventListener(...)</code>	Ajout direct
Une touche est pressée	<code>keydown</code>	<code>onkeydown</code>
Une touche est relâchée	<code>keyup</code>	<code>onkeyup</code>
Une touche de caractère est pressée	<code>keypress</code>	<code>onkeypress</code>
L'élément est chargé	<code>load</code>	<code>onload</code>
L'utilisateur sort de la page	<code>unload</code>	<code>onunload</code>
Le bouton de la souris est pressé	<code>mousedown</code>	<code>onmousedown</code>
Le bouton de la souris est relâché	<code>mouseup</code>	<code>onmouseup</code>
La souris est bougée	<code>mousemove</code>	<code>onmousemove</code>

Type d'événements

Descriptif	addEventListener(...)	Ajout direct
La souris survole un élément	mouseover	onmouseover
La taille de l'élément est réajustée	resize	onresize
Du texte est sélectionné	select	onselect

Fonctions sur les tableaux

array.map()

La méthode map() permet de créer un nouveau tableau sur base du résultat d'une fonction appelée sur chaque élément du tableau

array.map(callback)

callback(element[,index][,array])

Exemple:

```
t1 = [0,2,3,4];  
t2 = t1.map(function(e) {  
    return e * 2;  
});  
//t2 vaut [0,4,6,8]
```

array.reduce()

La méthode `reduce()` applique une fonction qui est un « accumulateur » et qui traite chaque valeur d'une liste (de la gauche vers la droite) afin de la réduire à une seule valeur.

`array.reduce(callback)`

`array.reduce(accumulateur,
valeurInitiale)`

`callback(element[,index][,array])`

Exemple:

```
const array1 = [1, 2, 3, 4];  
const reducer = (accumulator,  
  currentValue) => accumulator +  
  currentValue;  
  
// 1 + 2 + 3 + 4  
console.log(array1.reduce(reducer));  
// expected output: 10  
  
// 5 + 1 + 2 + 3 + 4  
console.log(array1.reduce(reducer,  
  5));  
// expected output: 15
```


LES FORMULAIRES

Objectifs des formulaires

- La création d'un formulaire se fait via la balise HTML `<form>`
- Les formulaires sont utilisés pour récolter des informations des utilisateurs
- 2 problèmes :
 - Comment faire transiter les données ?
 - Comment traiter les données reçues ?

Comment faire transiter les données ?

- 2 méthodes:
 - o GET: Fait transiter les données via l'adresse de la page
 - <https://www.monsite.com/page.php?cidReset=true&cidReq=B113B015>
 - Limite à 255 caractères
 - o POST: Fait transiter les données via la requête HTTP
 - Permet de faire transiter un plus gros nombre de caractères
- Définit avec l'attribut method
 - o `method="get"`
 - o `method="post"`

Comment faire transiter les données ?

- Il faut envoyer la requête contenant les données du formulaire (envoyé par GET ou POST) à un script qui pourra les traiter (ex. page contenant du PHP)
- Définit avec l'attribut action
 - o ex: action="./register.php"

Exemple de formulaire

Déclaration d'un formulaire

```
<form name="MyForm" method="get" action="./register.php">  
  <input type="text" name="nom">  
  <input type="text" name="prénom">  
  <input type="submit">  
</form>
```



Jean Dupont Envoyer

La balise <input>

- L'élément le plus important d'un formulaire est la balise <input>
- <input> est utilisé pour recueillir l'information de l'utilisateur
- <input> peut être utilisé de nombreuses manières différentes en fonction de la valeur de son attribut type

La balise <input>

Champs de texte

```
<input type="text" name="prenom">
```

Prénom:

Mot de passe

```
<input type="password" name="pwd">
```

Mot de passe :

Bouton radio

```
<input type="radio" name="sexe" value="male">Homme
```

```
<input type="radio" name="sexe" value="female">Femme
```

☒ Homme

☐ Femme

ATTENTION : La valeur de "name" doit être la même pour tous les boutons radios

La balise <input>

Checkbox

```
<input type="checkbox" name="type_musique" value="pop">Pop
```

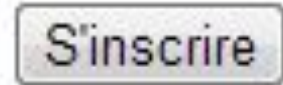
☐ Pop

```
<input type="checkbox" name="type_musique" value="rock">Rock
```

☐ Rock

Bouton submit

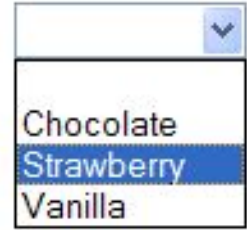
```
<input type="submit" value="S'inscrire">
```



La balise <select>

Select

```
<select name="voiture">
  <option value=""></option>
  <option value="choco">Chocolate</option>
  <option value="straw" selected>Strawberry</option>
  <option value="vanilla">Vanilla</option>
</select>
```



Possibilité de grouper les options

```
<optgroup label="Europe">
  <option value="straw" selected>Strawberry</option>
  <option value="vanilla">Vanilla</option>
</optgroup>
```



La balise <textarea>

Zone de texte

```
<textarea rows="10" cols="30">
```

Ceci est un texte déjà inscrit dans la zone de texte.

```
</textarea>
```

Commentaires:



L'attribut value

- Lorsque le formulaire est envoyé, ce sont les valeurs des attributs value qui sont envoyés au serveur liés au nom de l'input (name)
- Ecrire dans un input de type texte modifie son attribut value

La balise <label>

Permet de donner un libellé à un input

Peut être lié avec les attributs for et id

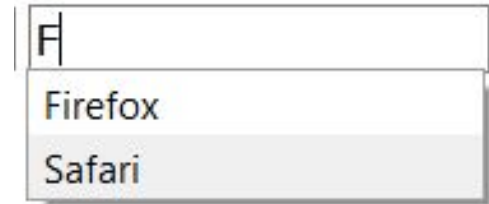
```
<label for="firstname">Prénom</label>
```

```
<input type="text" id="firstname" name="prenom">
```

La balise <datalist>

- La balise datalist spécifie une liste d'options prédéfinies pour un input
- Utile pour de l'autocomplétion (affichage d'une dropdown list)
- La datalist est liée à l'input via les attributs list et id

```
<input list="browsers">  
<datalist id="browsers">  
  <option value="Internet Explorer">  
  <option value="Firefox">  
  <option value="Chrome">  
  <option value="Opera">  
  <option value="Safari">  
</datalist>
```



Nouveaux types d'input – HTML5

- color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search
- tel (utile pour mobile (clavier adapté))
- time
- url
- week

**Attention à la
compatibilité avec
les navigateurs !**

Attributs de formulaire – HTML5

Nouveaux attributs pour `<input>`:

- `autocomplete`
- `autofocus`
- `checked` (checkbox, radio)
- `disabled`
- `form` (permet de désigner plusieurs forms)
- `formaction` (input)
- `formmethod` (input)
- `formnovalidate`

Attributs de formulaire – HTML5

Nouveaux attributs pour `<input>`:

- `formtarget`
- `height` et `width`
- `list`
- `min` et `max` (nombre, date)
- `maxlength` (nombre caractère max.)
- `multiple` (email, file)
- `pattern` (regexp)
- `placeholder` (exemple d'input de l'élément)
- `required`
- `value`
- ...

Objet Form du DOM

JavaScript permet de travailler sur les formulaires via un objet prédéfini Form

Il existe un objet Form pour chacun des formulaires d'un document HTML

Objet Form du DOM

Pour accéder aux formulaires:

```
document.forms[i];  
//i est la position du formulaire
```

```
document.forms["formName"];  
document.formName;  
//formName est le nom du formulaire
```

```
<form name="formName" method="get" action="./register.php">  
    ...  
</form>
```

Objet Form du DOM

```
const f = document.forms[i];  
//f est un objet form  
  
f.elements[];  
f[];  
//retourne un tableau contenant les éléments du  
formulaire  
  
f.reset();  
//vide le formulaire  
  
f.submit();  
//soumet le formulaire
```

Objet Element Input du DOM

```
let att = f[0].attributeName;  
//permet de récupérer la valeur d'un attribut d'un  
input  
  
f[0].attributeName = att;  
//permet de changer la valeur d'un attribut d'un  
input  
  
f[0].focus();
```

Exercice (1)

Créez un formulaire simple : Nom, Prénom.

À l'aide du JavaScript, proposez une complétion automatique du formulaire (Jean, Dupont) lorsque l'on appuie sur un bouton.

Rappels utiles :

```
const f = document.formName; // Accéder au formulaire
let valeur = f.champ.value; // Récupérer la valeur d'un champ
f.champ.value = "valeur"; // Insérer une valeur dans un champ
document.getElementById("id"); // Accéder à un élément de la page
```

Ajoutez un bouton qui remet à zéro tous les champs du formulaire

Conseil : Placez ces boutons hors du formulaire.

```
<button id="monBouton" >Auto-complétion</button>
```

Exercice (2)

Reprenez le formulaire précédent. Ajoutez-y un champ Code Postal.
À l'aide du JavaScript, faites les vérifications nécessaires sur les champs.
→ Un code postal est un nombre de 4 chiffres (entre 1000 et 9999).

Vérifiez si les champs sont bien remplis, et si le code postal est conforme.
Lorsque le formulaire est valide, un message s'affiche sur la page.

Rappels utiles :

```
let longueur = element.length; // Renvoie la longueur d'une chaîne de caractères  
let nbr = parseInt(variable); // Changer une chaîne de caractères en nombre  
element.innerHTML = "chaîne de caractères" // Ecrire dans un élément
```

LES EXPRESSIONS RÉGULIÈRES

Les expressions régulières

Une **expression rationnelle** ou **expression régulière** ou encore **RegEx** est en informatique une chaîne de caractères que l'on appelle parfois un motif et qui décrit un ensemble de chaînes de caractères possibles selon une syntaxe précise.

Objectifs:

Retrouver des patterns, des motifs, des structures dans une chaîne de caractères.

Exemple d'application

"Voici un petit texte qui contient plusieurs adresses email. Par exemple en voici une première: **toto@hotmail.com**. Si je vous présente cette adresse: **jules@gmail.com** vous pourrez également conclure qu'il s'agit d'une adresse email. Cependant **tata@msn** n'est pas une adresse valide. Il y a donc 2 emails dans cet exemple."

Une expression régulière va nous permettre d'extraire toutes les adresses mail.

Structure d'une expression régulière

/monExpressionRégulière/options

Exemple dans le code :

```
var regex = /[a-z]/gi;  
str = "ma chaîne de CarActèRes";  
str.match(regex);
```



ma chaîne de CarActèRes

Un outil pratique

Pour créer, tester et comprendre les expressions régulières :

<https://regex101.com/#javascript>

Pour visualiser des expressions régulières :

<https://regexper.com/>

Pour apprendre les expressions régulières :

<https://regexcrossword.com/>

Exprimer une chaîne de caractères

/chainedecaractères/options

Exemple :

Regex = /bonjour/i

"**Bonjour**, comment allez-vous ?"

Regex = /allez/i

"Bonjour, comment **allez**-vous ?"

Regex = /ou/gi

"Bonj**ou**r, comment allez-v**ou**s ?"

Exprimer une classe de caractères

Les crochets [] sont utilisés pour exprimer une classe de caractères

[xyz]	N'importe lequel des trois : x, y, ou z
[^xyz]	N'importe quel caractère sauf x, y, ou z
[a-z]	N'importe quel caractère de a à z minuscule
[^a-z]	N'importe quel caractère sauf une lettre minuscule de a à z
[a-fm-w]	N'importe quel caractère de a à f ou de m à w inclus
[^0-9A]	N'importe quel caractère sauf les chiffres de 0 à 9 ou la lettre A majuscule

Manipuler les expressions régulières

- Allez sur le site : <https://regex101.com>
 - Dans la zone de test, entrez ces phrases :
 - J'apprends le javascript
 - J'apprends le java
 - J'apprends le JavaScript
 - J'apprends le JAVASCRIPT

Écrivez la REGEX qui met en évidence le mot « javascript », peu importe la casse.

Comprendre une RegEx

- RegEx : `/gr[ioa]s/`
→ Quelles sont les possibilités de recherche avec cette regex ?

Autres opérateurs de REGEX

Caractère	Description/utilisation
.	Sélectionne n'importe quel caractère
\	Caractères spéciaux
^	Ne sélectionne que les occurrences en début d'input
\$	Ne sélectionne que les occurrences en fin d'input
*	Sélectionne le(s) caractère(s) entre 0 et n fois
+	Sélectionne le(s) caractère(s) entre 1 et n fois
?	Sélectionne le(s) caractère(s) entre 0 et 1 fois
x(?=y)	Sélectionne x seulement s'il est suivi de y
x(?!y)	Sélectionne x seulement s'il n'est pas suivi de y

Autres opérateurs de REGEX

Caractère	Description/utilisation
<code>x y</code>	Sélectionne x ou y (les deux)
<code>x{n}</code>	Sélectionne les occurrences de n x.
<code>x{n,m}</code>	Sélectionne les occurrences de n à m x.
<code>\s</code>	Sélectionne les espaces vides
<code>\d</code>	Sélectionne les caractères numériques (identiques à [0-9])
<code>\w</code>	Sélectionne tous les caractères alphanumériques et les underscores

Manipulons encore les RegEx

- Écrivez une RegEx qui permet de sélectionner une URL.
- Une URL se compose de :

`http://www.monsite.codePays`

Le code pays se compose de 2 à 3 lettres.

Trop facile ? ;-)

Essayez celle-ci : `http(s)://(www.)mon(.)site.codePays`

Parenthèses capturantes ou non

Les parenthèses permettent de capturer un sous-motif d'un motif

Exemple: Récupérer gmail et hotmail dans l'exemple précédent

```
[a-zA-Z0-9._- ]+@([a-z0-9._- ]{2,})\.[a-z]{2,4}
```

Pour utiliser des parenthèses non capturantes :

```
(?: ..... )
```

Recherche Gourmande/non Gourmande

- Par défaut, une RegEx recherchera la chaîne la plus longue possible
- Cependant, dans de nombreux cas, on est intéressé par l'occurrence la plus petite
- Exemple : Capturer une URL dans un ensemble de balises HTML

`//`

→ Valide pour : `Mon site`

→ Quid de : `<strong class="web">Mon site`

Recherche Gourmande/non Gourmande


- Solution:

`/<a href="(."+?)"/`

- Le point d'interrogation indique que la recherche doit s'arrêter dès qu'une chaîne correspondante (minimum) est trouvée

Créer un objet RegEx

```
let maRegExp = new RegExp("regex", "flags");  
let maRegExp = /regex/flags;
```



Les flags:

- Optionnels
- Peuvent prendre les valeurs:
 - i : non-sensible à la casse (maj/min)
 - g : sans le flag "g" la RegEx arrête l'analyse dès la découverte de la première occurrence
 - m : Multiligne. Affecte l'utilisation de ^ et \$ (recherche sur la ligne plutôt que sur les chaînes de caractères séparément)

Notation à privilégier

Exercice

- Pour être sûr de bien comprendre les RegEx, construisez les RegEx capablent de :
 1. Trouver tous les mots commençant par « fin ».
 2. Trouver tous les mots « cash » uniquement s'ils sont directement suivis de « flow(s) » (pas d'espace entre les deux mots)
 3. Trouver une RegEx capable d'extraire 800 millions et 15 milliards, c'est-à-dire :
[nombre]mill***

Les méthodes d'un objet RegEx

`reg.test(str)` : renvoie true si str vérifie la regex, false sinon

`reg.exec(str)` : applique la regex à la chaîne, renvoie le résultat

`str.match(reg)` : applique la regex à la chaîne, renvoie le(s) résultat(s)

`str.replace(reg, str2)` : remplace le(s) sous-chaîne(s) vérifiant la regex par str2 et renvoie le résultat

`str.search(reg)` : renvoie la position de la première sous-chaîne vérifiant la regex

`str.split(reg)` : pour "découper" une chaîne.

Derniers exercices

1. Dans l'exercice <http://jsbin.com/geqocoduzu/3/edit> , vérifiez que l'utilisateur a bien renseigné une adresse mail valide lorsqu'il clique sur le bouton « Valider ».
2. Idem 1, mais faites cette vérification en temps réel.
3. Dans l'exercice <http://jsbin.com/geqocoduzu/6/edit> , remplacez tous les montants en € en \$

Merci pour votre attention.

