HEH.be
Sciences
et technologies

Avenue V. Maistriau 8a
B-7000 Mons
+32 (0)65 33 81 54
scitech-mons@heh.be

WWW.HEH.BE

Année académique **2025 - 2026**

UE : Développement front-end

- **AA : Développement front-end – théorie**
  Ivan Miller
- **AA : Développement front-end – travaux pratiques**
  Ivan Miller & Baptiste Dambrin

Bachelier en Informatique
orientation réseaux et télécommunications

Ivan Miller
ivan.miller@heh.be

WALLONIE-BRUXELLES ENSEIGNEMENT
pôle hainuyer

Bienvenue dans le cours de développement front-end consacré aux fondamentaux du langage **JavaScript** et de **jQuery**. Ce cours combine des séances théoriques et pratiques et s'appuie sur deux outils en ligne :

- Sur l'**eCampus** (https://ecampus.heh.be), vous trouverez le syllabus, les fichiers nécessaires aux exercices, des exercices supplémentaires, les fichiers des démonstrations faites en classe, etc. C'est également par cette plateforme que vous remettrez le projet de fin d'année.
- Sur **Quizineur** (https://www.heh.be/quizineur), vous pourrez tester vos connaissances en vous exerçant sur des parcours QCM provenant d'une base de données de plus de 250 questions aux paramètres aléatoires.

Ce pictogramme signale des informations complémentaires, pour les plus téméraires d'entre vous. Les chances d'être interrogé sur cette matière sont très faibles, mais vous pourriez en avoir besoin plus tard ou lors de projets personnels.

AVERTISSEMENT : L'USAGE NON CRITIQUE DE L'INTELLIGENCE ARTIFICIELLE PEUT SABOTER VOTRE APPRENTISSAGE

L'IA peut sembler une aide précieuse. Elle peut aussi devenir un piège. Si vous vous reposez sur elle pour « aller plus vite » ou « éviter de réfléchir », vous ne développerez pas les compétences nécessaires à la réussite de ce cours.

Ivan Miller - ivan.miller@heh.be

Index

Chapitre 1 : JavaScript

1.1 Introduction au JavaScript

JavaScript est un langage web dynamique interprété qui s'intègre aux pages web afin d'en enrichir les fonctionnalités.

Son utilisation la plus courante consiste à surveiller les actions du visiteur et à y associer des traitements, qui se traduisent souvent par des modifications du corps de la page HTML.

Par exemple, lorsqu'un visiteur modifie le nombre d'articles dans son panier, le prix se met à jour ; ou lorsqu'un visiteur clique un bouton, une boîte de dialogue s'ouvre.

Derrière son apparente simplicité, JavaScript est un langage aux fonctionnalités riches, complexes et puissantes. Depuis les années 2000, il s'est imposé comme une technologie incontournable dans l'ensemble des métiers du Web.

Le succès du JavaScript le propage aujourd'hui à bien d'autres utilisations que les simples dynamismes de pages web auxquels il était destiné. Aujourd'hui, JavaScript est utilisé côté serveur avec Node.js, dans le développement de logiciels 3D, de jeux vidéo, d'applications Windows et mobiles, entre autres.

B. Historique

Les débuts

À l'origine, JavaScript était un projet de langage web côté serveur appelé « LiveScript », développé par Brendan Eich chez Netscape. À la fin de l'année 1995, il est adopté par la société Sun, également à l'origine du langage Java, et évolue alors vers une utilisation côté client.

Une décision marketing conduit ensuite à le renommer « JavaScript », entraînant de nombreuses confusions avec le langage Java, pourtant très différent. Java est un langage compilé, plus complexe et plus performant, principalement destiné au développement logiciel, tandis que JavaScript est interprété et orienté vers le Web.

ECMAScript

JavaScript est implémenté pour la première fois dans le navigateur Netscape en 1996. En réaction à cette nouvelle technologie, Microsoft développe son propre langage, nommé JScript.

Quelques mois plus tard, Netscape soumet JavaScript à l'Ecma International, un organisme européen de normalisation. Cette démarche aboutit à la publication, en 1997, de la première édition du standard **ECMAScript**, qui définit un ensemble de règles communes pour les langages de script tels que JavaScript, JScript et ActionScript.

En 1999, le standard connaît une évolution majeure avec ECMAScript Édition 3. Par la suite, le développement du langage rencontre certaines difficultés : la quatrième édition est finalement abandonnée. Une cinquième édition est toutefois publiée en décembre 2009, introduisant notamment la prise en charge du format **JSON**.

L'édition de juin 2015, connue sous le nom « **ES6** », marque une avancée importante en apportant de nombreuses nouveautés au langage. Depuis lors, une nouvelle version d'ECMAScript est publiée chaque année au mois de juin, proposant des améliorations mineures.

Succès du JavaScript

Si le succès du JavaScript était mitigé à ses débuts, plusieurs facteurs en font une technologie majeure dès le début des années 2000 : la standardisation du langage, son support par Internet Explorer 3.0 (1998), AJAX, l'arrivée des lignes ADSL...

Ce succès se poursuit avec l'arrivée de nouvelles technologies et outils : **jQuery** en 2006, **TypeScript** en 2012, **React.js** en 2013, ainsi que des milliers d'autres bibliothèques et plusieurs dizaines de frameworks.

C. Les outils nécessaires

Pour développer en JavaScript, il vous faut :

- Un **éditeur de code** comme Notepad++ ou Visual Studio Code
- Un **navigateur**
- Des connaissances en **HTML** et en **CSS**
- Des connaissances en **programmation**

1.2 Intégration du JavaScript

A. Comment l'intégrer ?

L'élément HTML *<script>* signale au navigateur les portions de code en JavaScript. Il peut soit contenir du code JavaScript interne, soit charger un fichier JavaScript externe :

```
<script> /* du JS */ </script>
<script src="js/monscript.js"></script>
</body>
```

Attention, un élément <script> ne peut pas charger du script interne et du script externe en même temps :

```
<script src="monscript.js"> /* du JS */ </script>
```

Mieux vaut éviter les attributs obsolètes *type="text/javascript"* et *language="javascript"* :

```
<script type="text/javascript" language="javascript"></script>
```

Utiliser des fichiers JavaScript externes permet de ne pas encombrer le code HTML et de réutiliser facilement les scripts sur plusieurs pages.

Lors du développement, il est recommandé d'utiliser **CTRL+F5** pour rafraîchir la page sans utiliser le cache du navigateur, afin de tester les dernières modifications du script.

En général, il est recommandé de ne pas ralentir l'interprétation du HTML en faisant passer le JavaScript avant. Pour cela :

- Codez les scripts avant la fermeture de </body>
- Utilisez les attributs booléens ***async*** ou ***defer***, en cas de scripts externes uniquement. Utilisez ***defer*** si l'ordre de vos fichiers JavaScript a une importance, sinon ***async***.

```
<script src="js/script.js" async></script>
<script src="js/autrescript.js" defer></script>
```

En dehors des éléments *<script>*, on retrouve du JavaScript dans certains attributs HTML comme *onclick* ou *onmouseenter* :

```
<div onclick="…" onmouseenter="…">
```

Cette syntaxe est toutefois déconseillée car, comme l'attribut *style* qui permet d'appliquer du CSS à un élément HTML, elle nuit à la lisibilité de votre code en mélangeant les différents langages.

B. <noscript>

```
<noscript>
<p>Ce code HTML n'est interprété que si JavaScript est désactivé.</p>
</noscript>
```

C. Les commentaires

JavaScript utilise les mêmes conventions que le PHP et le C++ pour les commentaires :

```
<script>
//commentaire jusqu'en fin de ligne
/* commentaire
sur plusieurs lignes */
</script>
```

D. Masquer les scripts pour les anciens navigateurs

Dans les années 90, alors que certains navigateurs antiques (IE 1, IE 2 et Netscape 1) ne comprenaient pas le JavaScript, il était de bon usage de commenter les scripts avec la syntaxe HTML < !-- … -->, pour éviter que le script soit confondu avec du simple texte et affiché dans la page.

```
<script>
<!--
…
-->
</script>
```

De nos jours, le JavaScript est supporté par tous les navigateurs et nous pouvons nous passer de cette habitude.

E. Le point-virgule ;

En JavaScript, le point-virgule permet de séparer les instructions, au même titre que le retour à la ligne. Malgré les différentes syntaxes possibles, il est vivement recommandé de terminer chaque instruction par un point-virgule.

Si vous avez déjà codé en langage C ou en PHP, langages où le point-virgule sert de terminateur d'instruction plutôt que de simple séparateur, sachez que JavaScript est plus flexible : le point-virgule n'est pas strictement obligatoire, même s'il reste recommandé.

F. Les accolades { }

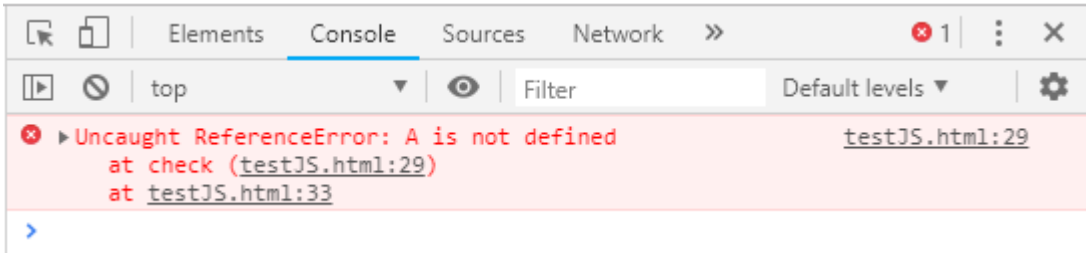
En JavaScript, le regroupement d'instructions se fait en les délimitant par des accolades { }. Ce regroupement s'appelle un « bloc d'instructions ».

Le nombre d'accolades ouvrantes doit toujours être égal au nombre d'accolades fermantes.

G. Débuguer et utiliser la console

Pour débuguer un code JavaScript récalcitrant, n'oublions pas les méthodes classiques comme la relecture du code, la mise en commentaire de parties de code ou l'affichage de messages à des endroits stratégique.

Les outils de développement des navigateurs sont équipés d'une console JavaScript mentionnant divers messages d'erreurs



Mais vous pouvez aussi exploiter cette console pour y consigner vos propres messages, par utilisation de l'objet *console* :

```
console.log("Message console") ;
```

Parmi les nombreuses méthodes de l'objet console, voici les principales :

`console.count(x)` : un compteur par chaîne *x*
`console.dir(objet)` : une liste des propriétés d'un objet
`console.error("Message d'erreur")` : un message d'erreur
`console.log(variable)` : du texte ou des contenus de variables
`console.table(tableau)` : le contenu d'un tableau
`console.time("chrono1")` puis `console.timeEnd("chrono1")` : chronomètre
`console.warn("Message d'avertissement")` : un message d'avertissement

H. Conseils pour un code robuste et actuel

Si vous souhaitez un code robuste et actuel, évitez :

- De coder des vieux attributs *type="text/javascript", language="javascript"* dans l'élément *<script>*
- De coder des commentaires HTML *< !-- -->* dans les scripts
- D'utiliser la fonction *eval()*
- D'utiliser *with()*
- D'utiliser *document.write()*
- De déclarer les variables de manière implicite
- De terminer les instructions par un retour à la ligne au lieu d'un « ; »

1.3 Variables

Les **variables** sont des conteneurs de données. Le processeur les repère par leurs adresses, alors que le programmeur les repère par leurs noms. Le contenu d'une variable peut être de **type primitif** (nombre, chaîne, booléen, indéfini) ou de **type objet**.

A. Déclarations de variables

La déclaration d'une variable est une instruction provoquant la réservation d'un espace mémoire et l'attribution de son nom de variable.

Voici 2 exemples de **déclarations** :

```
var a ;

let b ;     // syntaxe + actuelle
```

L'utilisation de **let** est préférable car sa portée est inférieure à celle de **var**.
Il est possible de déclarer plusieurs variables en une fois :

```
let c, d, e ;
```

Une déclaration peut s'accompagner d'une 1re affectation, appelée **initialisation** :

```
let f = 28 ;

let g = h = 42 ;
```

Les variables déclarées sans valeur initiale valent ***undefined***.

Une fois déclarée, la variable peut changer de valeur, c'est l'opération d'**affectation** :

```
d = 48.5 ;
```

Les **constantes** sont des variables dont la valeur est non modifiable après initialisation :

```
const c = 400 ;
```

L'affectation d'une variable non déclarée provoque sa déclaration. Cette syntaxe, fortement déconseillée, est appelée « **déclaration implicite** » :

```
i = 33 ;
```

Au contraire, utiliser une variable non déclarée afin d'en demander la valeur, provoque une **erreur** affichée dans la console du navigateur :

```
alert(k) ;        // Uncaught ReferenceError: k is not defined
```

B. Portées des variables

La façon de déclarer une variable a un impact sur sa **portée**.
La « portée » d'une variable (*scope* en anglais) désigne les parties du code où la variable est accessible. La portée peut être **globale** (accessible partout) ou **locale** (limitée).

Une variable locale est détruite à la fin de sa portée, ce qui a pour effet de libérer son espace mémoire et son nom. La variable globale n'est détruite qu'à la fermeture de la page. Les variables globales ont donc tendance à surcharger la mémoire mais aussi à provoquer d'éventuelles erreurs de redondance de noms, ce qui freine la réutilisation de vos codes.
Les variables globales ne sont pas interdites, mais à éviter.

L'utilisation du mot-clé ***let*** dans un bloc d'instructions, permet de déclarer une variable locale à ce bloc d'instructions. Un bloc d'instructions est une série d'instructions comprises entre accolades { }. En dehors de tout bloc d'instructions, le mot-clé ***let*** permet de déclarer des variables globales.

```
for (let i = 0 ; i<10 ; i++ ) {
      console.log(i);   //affiche la valeur de i dans la console
}
console.log(i);   //provoque une erreur
```

L'utilisation du mot-clé ***var*** dans une fonction, permet de déclarer une variable locale à cette fonction. En dehors d'une fonction, le mot-clé ***var*** permet de déclarer des variables globales. Ces variables globales déclarées avec ***var*** ont la particularité de s'ajouter comme propriété de l'objet *window*.

```
var x = 8;
console.log(window.x);  // 8
console.log(x);         // 8
let y = 9;
console.log(window.y);  // undefined
console.log(y);         // 9
```

Enfin, une variable déclarée implicitement, c'est-à-dire sans mot-clé, est globale, quel que soit l'endroit où elle est déclarée. Rappelons que les variables globales sont à éviter au maximum pour cause de surcharge inutile de la mémoire et de risques de doublons de noms de variables.

Exercice écrit : portée des variables

Quelles portées ont les variables suivantes ?

```
var A;
let B;
C = 2 ;
function check() {
      var D;
      let E;
      F = 2;
      if (true) {
            var G;
            let H;
            I = 2;
      }
}
check();
```

C. Noms de variables

Le choix d'un nom de variable doit respecter les règles suivantes :

- Le nom doit commencer par une lettre, un "_" ou un "$" ;
- Le nom doit se composer uniquement de lettres (y compris lettres accentuées), de chiffres et des caractères "_" et "$" ;
- Les caractères accentués ne peuvent pas être suivis d'un chiffre ;
- Le JavaScript est sensible à la casse (attention aux majuscules) ;
- Le nom ne peut pas être un nom réservé dont voici la liste :

abstract	boolean	break	byte	case
catch	char	class	const	continue
default	do	double	else	extends
false	final	finally	float	for
function	goto	if	implements	import
in	instanceof	int	interface	long
native	new	null	package	private
protected	public	return	short	static
super	switch	synchronized	this	throw
throws	transient	true	try	typeof
var	void	while	with	

À ces interdictions, s'ajoutent des noms à éviter pour les variables globales : ***document, history, location, navigator, top, window, …***

Conseils : choisissez toujours des noms de variables représentatifs de leur contenu et choisissez une écriture commune à toutes vos variables :

- ageClient : *camelCase* ou écriture en dos de chameau
- age_client : écriture avec *underscore* à la place des espaces

D. Types primitifs

Les variables peuvent contenir des objets ou des données de types primitifs. Une variable inexistante est dite ***null.*** *Null est une* valeur primitive.

JavaScript propose 4 types primitifs de données :

- Les **nombres** : entiers ou avec virgule tels que 45 ou 3.1416 ; le nombre peut commencer par 0 pour une écriture octale ou par 0x pour une écriture hexadécimale.
- Les **chaînes de caractères** (*strings*) : c'est-à-dire des suites de caractères entre guillemets ou entre apostrophes, au choix, telles que "Isabelle" ou 'Émilie'.
- Les **booléens** : qui valent *true* pour vrai ou *false* pour faux.
- Les **indéfinis** (ou ***undefined***) : les variables non affectées. Exemple : `let bidule ;`

Le JavaScript est un langage à **typage dynamique** : le type des variables peut changer en cours d'exécution. Il est donc possible d'affecter un nombre à une variable, puis d'y affecter une chaîne de caractères ou un booléen. Exemple :

```
x = 8 ;
x = false ;
x = "Barnabé" ;
```

En cas de doute, l'opérateur ***typeof*** permet de connaître le type d'une variable :

```
console.log( typeof x ) ;
```

Certaines erreurs de programmation liées aux types peuvent survenir…

Exemple d'erreur :

```
let a = "gladi";
let b = "ateur";
console.log (a*b);      //affiche NaN
```

Multiplier des chaînes de caractères est une erreur. C'est pourquoi, ce script va afficher l'expression « **NaN** » signifiant « **Not a Number** ».

Autre exemple d'erreur :

```
let z = 8 ;
console.log(z.length) ;       //affiche undefined
```

length est une propriété des chaînes de caractères. Dans cet exemple, *z* ne contient pas une chaîne de caractères et ne peut donc pas accéder à sa propriété *length.* Ce script va afficher « **undefined** ».

E. Conversions de types

Pour éviter certaines erreurs liées au type des variables, nous avons à disposition une série de fonctions de conversion.

`parseInt(x)`	Convertit x en nombre entier
`parseInt(x,b)`	Convertit x en nombre entier décimal avec x écrit en base b
`parseFloat(x)`	Convertit x en nombre réel
`Number(x)`	Convertit x en nombre
`String(x)`	Convertit x en chaîne de caractères
`Boolean(x)`	Convertit x en booléen

Autres conversions possibles :

`x.toString()`	Méthode convertissant x en chaîne de caractères
`x.toFixed(`*n*`)`	Méthode arrondissant un nombre *x* à *n* décimales et convertissant en chaîne de caractères
`s*1`	Multiplier la chaîne de caractères *s* par 1 donne un nombre
`s+''`	Concaténer *s* avec une chaîne vide donne une chaîne de caractères

F. Emploi des guillemets et des apostrophes

En JavaScript, les valeurs de type **chaîne de caractères** sont délimitées par des **guillemets** ou des **apostrophes**, au choix :

```
"Je suis une chaîne"
'moi aussi !'
```

Dans une chaîne de caractères délimitée par des apostrophes, l'utilisation d'une apostrophe non échappée provoque une erreur. Il en va de même pour les guillemets dans une chaîne encadrée par des guillemets. Pour éviter ce problème, il faut échapper ces caractères en les faisant précéder du **caractère d'échappement « \ »** :

```
let com1 = '<p class="top">C\'est trop bien !</p>' ;
let com2 = "<p class=\"top\">C'est trop bien !</p>" ;
```

Utiliser un nom de variable dans une chaîne ne valorise pas la variable : le résultat est le nom de la variable, pas sa valeur.

```
let x = 5 ;
console.log("J'ai x chats !") ;       // résultat : J'ai x chats !
```

Pour valoriser une variable, elle doit se trouver hors de la chaîne :

```
let x = 5 ;
console.log("J'ai "+ x +" chats !") ; // résultat : J'ai 5 chats !
```

Mais il existe aussi une syntaxe ES6, appelée « **gabarits** », permettant de contenir une chaîne entre accents graves tout en valorisant les variables grâce à la syntaxe ${} :

```
let x = 5 ;
console.log(`J'ai ${x} chats !`) ;   // résultat : J'ai 5 chats !
```

G. Méthodes et propriétés des *strings*

Comment un type primitif comme *string* peut-il avoir des méthodes et des propriétés comme un objet ? Parce que le type primitif *string* est encapsulé dans l'objet *String* proposant ces méthodes et propriétés. Lors de l'utilisation d'une de ces méthodes/propriétés, le *string* est temporairement converti en objet *String*.

Soit « a » une chaîne de caractères :

a.charAt(0) **a.charAt(8)**	Donne le 1er caractère de la chaîne *a* Donne le 9e caractère de la chaîne *a*
a.endsWith(z)	Teste si une chaîne se termine par la chaîne *z*
a.indexOf('x')	Donne la position de la première occurrence de la lettre « *x* » dans la chaîne *a* ; ou -1 si pas d'occurrence.
a.lastIndexOf('x')	Donne la position de la dernière occurrence de la lettre « x » dans la chaîne *a* ; ou -1 si pas d'occurrence.
a.length	Contient le nombre de caractères de la chaîne *a*
tab = a.split('-')	Saucissonne une chaîne selon les occurrences de la chaîne passée en paramètre, afin de générer un tableau.
a.substring(4,10) **a.substring(4)**	Donne les caractères de la chaîne *a* compris entre le 5e et le 11e. Donne les caractères de la chaîne *a* depuis le 5e jusqu'à la fin.
a.substr(4,10)	Donne les caractères de la chaîne *a* compris entre le 5e et le 15e
a.toLowerCase()	Donne la chaîne réécrite en minuscules
a.toUpperCase()	Donne la chaîne réécrite en majuscules
a.replace('PHP','JS')	Renvoie la chaîne *a* avec « PHP » remplacé par « JS »

Remarque : la chaîne *a* n'est pas impactée par l'utilisation de ces méthodes. Pour modifier la chaîne *a*, l'appel de la fonction ne suffit pas, il faut aussi affecter *a* :

```
a = a.toLowerCase();
```

1.4 Tableaux

Les tableaux permettent de stocker plusieurs valeurs dans une même entité. Chaque valeur ainsi stockée se voit attribuer un indice afin d'y accéder.

```
tab[0] = 42 ;  // qui est le tableau ? l'indice ? la valeur ?
```

En JavaScript, les tableaux sont des objets **Array**.

Il existe deux types de tableaux :

- **Tableaux indicés :** les indices sont des nombres entiers.
- **Tableaux associatifs :** les indices sont des chaînes de caractères.

A. Tableaux indicés

Déclaration d'un tableau de 10 cases

```
let tableau = new Array(10);
```

Initialisations de tableaux indicés

```
let tableau = new Array("ok", 15, 7 , 'JS') ;
let variante = ["ok", 15, 7 , 'JS'] ;
let tab = [7] ; //Déclaration d'un tableau d'1 valeur (valant 7)
```

Affectations

```
jours[0] = "lundi";
jours[1] = "mardi";
…
```

Parcourir un tableau indicé « jours »

```
for (let i = 0 ; i < jours.length ; i++) {
      concole.log(jours[i]) ;
}
```

Remarque : comme les tableaux sont des objets, on peut en exploiter les propriétés et les méthodes. Ici, « **length** » est une propriété du tableau « jours » et permet de connaître aisément le nombre de valeurs contenues dans ce tableau.

Remarque : les indices des tableaux sont bien des propriétés. Pourtant, on ne peut pas accéder à ces propriétés en utilisant la syntaxe *objet.propriété*, il faut utiliser la syntaxe *objet[propriété].* Ceci est d'application pour toutes les propriétés dont le nom commence par un chiffre, appelées **propriétés énumérables**.

B. Tableaux associatifs

Déclaration :

```
let navig = { "Firefox" : 8, "Chrome" : 51, "Safari": 19};
```

Exemple d'affectations :

```
navig["Firefox"] = 9;
navig["Chrome"] = 52;
navig["Safari"] = 21;
```

Parcourir un tableau associatif

```
for (let i in navig) {
      console.log(navig[i]) ;
}
```

Remarque : Les tableaux associatifs peuvent être utilisés comme des tableaux indicés afin de récupérer ou d'utiliser leurs indices.

C. Tableaux indicés à plusieurs dimensions

Déclaration et affectation d'un tableau de 2 lignes et 2 colonnes :

```
let tableau = new Array();    // déclaration d'un tableau à 1 dimension
tableau[0] = new Array();     // hop, le 1er élément devient un tableau
tableau[1] = new Array();     // hop, le 2e élément devient un tableau

tableau[0][0] = "Julien";     // exemples d'affectations
tableau[0][1] = 32;
tableau[1][0] = "Quentin";
tableau[1][1] = 26;
```

Variante où l'on colle 2 tableaux à 1 dimension en un tableau à 2 dimensions :

```
let aNoms = new Array("Nom1", "Nom2", "Nom3");
let aPrenom = new Array("Prénom1", "Prénom2", "Prénom3");
let monTableau = new Array(aNoms, aPrenom);
```

Variantes avec initialisation :

```
let tab1 = new Array(["Bob", "Luc", "Joe"], ["Bill", "Luca", "Jim"])
let tab2 = [[25, 2], [6, 8], [3, 18]];
```

D. Quelques méthodes et propriétés de l'objet Array

Soit « tableau » un objet *Array* :

tableau.concat(tab2)	Concatène le tableau tab2 à la fin de tableau.
tableau.includes(x)	Teste si la valeur x est contenue dans le tableau et apporte une réponse booléenne.
tableau.indexOf(x)	Renvoie la position de la 1re occurrence de la valeur *x* dans le tableau, ou -1 si la valeur ne s'y trouve pas.
tableau.join(' ; ')	Regroupe tous les éléments du tableau en une seule chaîne de caractères avec comme séparateur ' ; '
tableau.length	Contient le nombre d'éléments du tableau.
tableau.pop()	Enlève le dernier élément du tableau.
tableau.push(x)	Ajoute x en fin de tableau.
tableau.reverse( )	Inverse l'ordre des éléments.
tableau.shift( )	Enlève le premier élément du tableau.
tableau.sort()	Trie les éléments d'un tableau par ordre croissant.
tableau.splice(pos, n)	Supprime *n* éléments à partir de la position *pos.*
tableau.unshift(x)	Ajoute x au début du tableau (et décale le reste).

E. Parcourir un tableau avec la méthode forEach()

La méthode *forEach()* parcourt les valeurs d'un tableau et appelle une fonction pour chacune.

```
function maFonction(x) {
      console.log(x);
}
let tab = new Array("Pierre", "Jean", "Jacques");
tab.forEach(maFonction);
```

Syntaxe avec une fonction fléchée (expliquée plus loin dans le cours) :

```
let tab = new Array("Pierre", "Jean", "Jacques");
tab.forEach(val => console.log(val));
```

F. Vérifier la présence d'un indice dans un tableau avec *in*

L'opérateur ***in*** vérifie la présence d'un indice dans un tableau (ou plus largement, d'une propriété dans un objet) :

```
let fleurs = new Array("bégonia", "dahlia", "iris", "magnolia");
3 in fleurs       // true : le tableau possède un indice 3
"iris" in fleurs  // false : « iris » est une valeur
length in fleurs // true
```

1.5 Opérateurs

Affectation =

Opérateurs arithmétiques

Addition	+	
Soustraction	-	
Multiplication	*	
Exponentiation	**	
Division Entière	/	Uniquement sur des types entiers
Modulo	%	Reste de la division entière
Division Réelle	/	Uniquement sur des types réels

Opérateurs d'incrémentation

Incrémentation	`++i ;`	ou	`i++ ;`
Décrémentation	`--i ;`	ou	`i-- ;`

Opérateurs d'affectation élargie (raccourci d'une affectation et d'une opération)

`i += k ;`	équivaut à	`i = i + k ;`
`i -= k ;`	équivaut à	`i = i - k ;`
`i *= k ;`	équivaut à	`i = i * k ;`
`i /= k ;`	équivaut à	`i = i / k ;`
`i %= k ;`	équivaut à	`i = i % k ;`

Opérateur de concaténation

Concaténation	+	Uniquement si l'une des valeurs est une chaîne de caractères

Opérateurs logiques

Et	&&
Ou	||
Contraire	!

Opérateurs relationnels (ou de comparaison)

Plus grand que	>
Plus petit que	<
Plus grand ou égal à	>=
Plus petit ou égal à	<=
Égal à	==
Égal à (type et valeur)	===
Différent de	!=

Remarque
= est un opérateur d'affectation (changement de la valeur) à ne pas confondre avec == qui est un opérateur de comparaison (souvent utilisé dans les conditions).

JavaScript dispose également de 7 opérateurs binaires pour les opérations sur les valeurs binaires : & | ^ ~ << >> >>>

1.6 Structures conditionnelles

Les structures conditionnelles sont des tests permettant d'exécuter ou non une série d'instructions.

A. *if*

L'instruction *if* conditionne l'exécution d'une instruction. La condition est une expression pouvant s'avérer vraie ou fausse. Si la condition est vraie, l'instruction est exécutée. L'instruction *if* peut être suivie d'une instruction *else* (sinon) présentant une instruction à exécuter au cas où la condition est fausse.

```
if (condition) instruction1 ;
else instruction2 ;
```

Une condition n'a que deux résultats possibles : vrai ou faux.
Si la condition est vraie, l'instruction 1 est exécutée, si la condition est fausse, l'instruction 2 est exécutée.

Remarques :

- La condition s'écrit toujours entre parenthèses.

- Les accolades servent à délimiter des blocs d'instructions. Une instruction seule peut s'en passer, mais s'il y a plusieurs instructions, les accolades deviennent obligatoires :

```
if (condition) {
      instruction1 ;
      instruction2 ;
}
```

- Il est possible de poser une seconde condition si la première est fausse, comme suit :

```
if (condition1) {
      instruction1 ;
} else if (condition2) {
      instruction2 ;
} else {
      instruction3 ;
}
```

- Il est possible de poser des conditions multiples liées par les opérateurs logiques AND (&&) ou OR (||) :

```
if ((condition1) && (condition2)) instruction ;
```

B. *switch*

L'instruction *switch* permet de faire plusieurs tests de valeurs sur le contenu d'une même variable.

Exemple de syntaxe :

```
switch (variable) {
      case 1 :    instructions1;
                  break;
      case 2 :    instructions2;
                  break;
      case 3 :    instructions3;
                  break;
      default : instructions_par_défaut;
}
```

Les parenthèses qui suivent le mot clé ***switch*** indiquent une variable dont la valeur est testée successivement par chacun des ***case***. Lorsque la variable testée est égale à une des valeurs suivant un ***case***, la liste d'instructions qui suit est exécutée.

Le mot clé ***break*** indique la sortie de la structure conditionnelle.
Le mot clé ***default*** précède les instructions qui seront exécutées si l'expression n'est égale à aucune des valeurs proposées dans les ***case***.

Remarques : En termes d'optimisation des performances de votre code JavaScript, il est conseillé d'opter pour un *switch* à partir de 3 conditions, autrement une structure *if/else* est conseillée.
Habituellement, on énumère les conditions du *switch* par ordre croissant, par souci de lisibilité. Cependant, il ne s'agit pas d'une obligation et, point de vue performances, comme pour les structures *if/else*, il est conseillé de toujours commencer par la condition qui sera la plus souvent vérifiée.

C. Opérateur ternaire « *?* »

L'opérateur ternaire peut dans certains cas remplacer une structure *if/else*. Ses possibilités sont plus limitées, mais son exécution est plus rapide et sa syntaxe plus réduite :

```
(age>=18) ? acces=true : acces=false ;
```

Ou encore :

```
let acces = (age>=18) ? true : false ;
```

1.7 Structures itératives

Les structures itératives, également appelées boucles, permettent de répéter des instructions.

A. *while*

```
while (condition) {
      instructions;
}
```

Cette instruction exécute les instructions **tant que** (*while* signifiant « tant que ») la condition est vraie.

Le risque est que la condition reste vraie… toujours ! On appelle cela une **boucle infinie**, c'est-à-dire un plantage du programme qui n'arrive jamais à sortir de la boucle en question.

B. *do while*

L'instruction ***do while*** est semblable à l'instruction *while* à ceci près que la condition se trouve en fin de boucle. Le test de la condition ayant lieu après les instructions, celles-ci sont exécutées au moins une fois par le programme (même si la condition est fausse).

Cette boucle est conseillée lorsque la condition porte sur une calculée durant la boucle et non initialisée.

La syntaxe est la suivante :

```
do {
      instructions;
 } while (condition);
```

C. *for*

La troisième structure itérative est l'instruction ***for***. Comme pour l'instruction *while*, la condition se fait avant les instructions, mais cette fois avec une écriture condensée comprenant l'affectation du compteur, la condition et la progression (incrémentation par exemple) du compteur.

La syntaxe est la suivante :

```
for (affectation ; condition ; progression) {
      instructions;
}
```

D. *for in*

Variante du *for*, le ***for in*** permet d'exploiter tous les indices des propriétés énumérables d'un objet ou d'un tableau.

L'avantage de cette boucle est qu'elle ne nécessite pas de connaitre les noms ou le nombre de propriétés d'un objet.

L'inconvénient est que l'ordre dans lequel sont parcourues les propriétés énumérables de l'objet est imprévisible. L'usage de cette boucle est donc à proscrire si l'ordre dans lequel vous parcourez votre tableau a une importance.

Exemple :

```
for (let prop in mon_objet) {
  console.log(prop + " = " + mon_objet[prop]) ;
}
```

E. *for of*

Autre variante, le ***for of*** permet d'exploiter toutes les valeurs énumérables d'un objet (voir toutes les valeurs d'un tableau, puisque les tableaux sont des objets).

Exemple :

```
for (let val of mon_objet) {
  console.log("valeur : " + val) ;
}
```

Appliqué à une chaîne de caractères, le *for of* parcourt les caractères de la chaîne un par un.

F. *break* et *continue*

L'instruction ***break*** permet d'interrompre prématurément une boucle ou la structure conditionnelle *switch* (mais pas un *if*).

L'instruction ***continue***, dans une boucle, permet de sauter à l'itération suivante.

1.8 Fonctions

Les fonctions sont des blocs de code réutilisables.
En codant vos scripts, identifiez les parties redondantes et isolez-les dans des fonctions. Vous vous constituerez ainsi une batterie d'outils disponibles en un simple appel.

Les fonctions présentent 2 intérêts majeurs en JavaScript : éviter la redondance de code et libérer la mémoire en détruisant ses variables locales.

A. Déclaration de fonctions

La déclaration d'une fonction se fait par l'utilisation du mot-clé ***function***. Exemple :

```
function nom_de_la_fonction (paramètre1, paramètre2, …) {
      //instructions de la fonction
}
```

Le nom de la fonction doit respecter les mêmes règles que les noms de variables.

Déclarer une fonction n'entraine pas son exécution. L'exécution des fonctions n'est entraînée par leurs **appels**. Exemple :

```
nom_de_la_fonction(argument1, argument2, …);
```

La déclaration d'une fonction peut être faite avant ou après son appel, car le code JavaScript prend connaissance des déclarations de fonction lors de sa pré-lecture par le navigateur.

Les **paramètres** sont facultatifs, mais les parenthèses sont obligatoires lors de la déclaration.

Le nom de la fonction est aussi facultatif : une fonction sans nom est une **fonction anonyme**.

Certaines fonctions renvoient un résultat (une valeur). Ce « retour » est signalé par le mot-clé ***return*** qui met fin à la fonction :

```
function aire_triangle(longueur, largeur) {
      let aire = (longueur * largeur) /2 ;
      return aire;
}
```

Voici un exemple d'appel de cette fonction passant les valeurs 5 et 4 en arguments et attribuant le résultat renvoyé à la variable *surface* :

```
let surface = aire_triangle(5, 4) ;
```

B. Répétition de noms de variables/fonctions

Voici un extrait de code JavaScript mettant en évidence la possibilité d'utiliser plusieurs fois le même nom de variable ou de fonction, tant que leurs portées diffèrent.

```
function aire(longueur, largeur) {
      let aire = longueur * largeur ;
      return aire;
}
let longueur = 5, largeur = 4 ;
let surface = aire(5, 4) ;
```

Dans cet exemple, la fonction *aire* globale et la variable *aire* locale à cette fonction portent le même nom.

Toujours dans cet exemple, les variables *longueur* et *largeur* locales à la fonction *aire* et les variables *longueur* et *largeur* globales portent le même nom alors qu'il s'agit bien de 4 variables différentes.

C. Arguments et paramètres

On appelle « **paramètres** » les variables présentes dans les parenthèses de la définition de la fonction. On appelle « **arguments** » les valeurs passées à la fonction lors de son appel, valeurs servant à initialiser les paramètres.

Dans l'exemple ci-dessous, la fonction *message()* est appelée et reçoit en argument la valeur « Bienvenue » qui initialise le paramètre *texte*.

```
function message(texte) {
      alert("Message : " + texte);
}
message("Bienvenue");   // appel de la fonction
```

Une fonction peut avoir jusqu'à 255 paramètres.

Depuis ES6, il est possible de donner une valeur par défaut à un paramètre (afin par exemple de s'assurer de son existence) :

```
function message(texte = 'Coucou') {
      alert("Message : " + texte);
}
message();  // appel de la fonction sans argument
```

D. Expressions de fonction

```
let coucou = function () {
    alert('Coucou !');
}
```

Une **expression de fonction** est une déclaration de fonction affectée à une variable.
Les expressions de fonctions, contrairement aux déclarations de fonctions, ne peuvent pas être codées après leurs appels.

```
coucou(); // appel ok 😊
function coucou() {
    alert('Coucou');
}
coucou(); // appel ok 😊
```

```
bonjour(); // erreur ☹
let bonjour = function() {
    alert('Bonjour');
}
bonjour(); // appel ok 😊
```

E. Fonctions anonymes

```
function () {
      // instructions de la fonction
}
```

Il est possible de déclarer des fonctions sans nom, dites **fonctions anonymes** :

- Leur usage est unique : sans nom, impossible de les appeler à nouveau.
- Leur intérêt est principalement de libérer les espaces mémoires utilisés par les éventuelles variables locales.

Comment exécuter une fonction si nous ne pouvons pas l'appeler ? JavaScript nous propose 3 possibilités :

- **L'expression de fonction** : Affectée, la fonction anonyme peut être appelée en suivant le nom de la variable d'une paire de parenthèses « appelantes » :

```
let bidule = function () {
   // instructions de la fonction
}
bidule() ;
```

- **L'auto-invocation** : en mettant la déclaration de la fonction anonyme entre parenthèses et en la suivant d'une paire de parenthèses « appelantes » :

```
( function () {
      // instructions de la fonction
} ) () ;
```

- **L'écoute d'évènement** pour déclencher l'exécution de notre fonction :

```
truc.addEventListener('click', function() { alert('Hop'); });
```

F. Fonctions fléchées

En 2015, ES6 introduit une nouvelle syntaxe de déclaration des fonctions anonymes appelée fonctions fléchées ou *Arrow functions* :

```
x => x + 3 ;      // 1 paramètre
(x, y) => x * y;  // 2 paramètres
```

Ou, écriture plus explicite :

```
(x, y) => { return x * y; } ;
```

Ce qui équivaut presque à :

```
function(x, y) {
    return x * y;
}
```

G. Map

La méthode *map()* génère un tableau en appliquant une fonction à tous les éléments d'un tableau.

```
let nombres = [12, 66, 5, 40];
let nouveauTableau = nombres.map(maFonction);

function maFonction(num) {
  return num + 10;
}
```

H. Closures

Une closure est une fonction interne qui va « se souvenir » et pouvoir continuer à accéder à des variables définies dans sa fonction parente même après la fin de l'exécution de celle-ci.

Exemple :

```
function compteur() {
    let count = 0;
    return function() {
          return count++;
    };
}
let plusUn = compteur();
```

I. Les fonctions asynchrones

Le JavaScript est démuni d'instruction marquant une pause dans l'interprétation du code. Il est néanmoins possible d'appeler une fonction après un laps de temps déterminé en millisecondes. On parle alors de fonctions asynchrones.

Exemple d'appel de fonction après 1000ms ou toutes les 1000ms (intervalle) :

```
setTimeout(nomFonction, 1000);
setInterval(nomFonction, 1000);
```

Syntaxe si des paramètres sont passés à la fonction :

```
setTimeout(function() { nomFonction(x); }, 1000);
setInterval(function() { nomFonction(x); }, 1000);
```

Un intervalle peut être arrêté comme suit :

```
const monInterval = setInterval(nomFonction, 1000);
clearInterval(monInterval) ;
```

Une fonction asynchrone ne bloque pas le reste du script et l'attente de sa réponse est appelée « **promesse** ».

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}
async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
}
asyncCall();
```

J. Résultat ou référence de fonction ?

Dans le point précédent, nous rencontrons 2 syntaxes : l'une accompagnée de parenthèses, l'autre non. Cela a son importance.

La syntaxe suivante affecte le résultat de la fonction à la variable x :

```
let x = nomFonction() ;
```

La syntaxe suivante affecte la référence (adresse) de la fonction à la variable x :

```
let x = nomFonction ;
```

1.9 Objets

A. Les objets

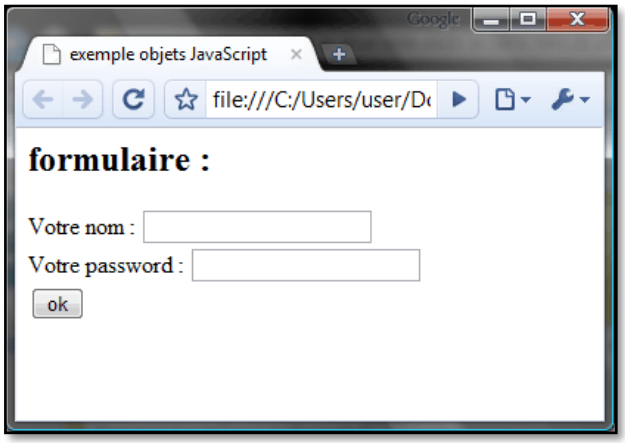
JavaScript est un langage **orienté objet.** Un objet est un ensemble de propriétés regroupées dans une même entité. La valeur d'une propriété peut être une fonction, dans ce cas la propriété est appelée « méthode ».

Chaque objet JavaScript contient des méthodes (fonctions propres à l'objet) et des propriétés (variables propres à l'objet). La syntaxe est la suivante :

```
objet.methode()
objet.propriete
```

Exemples :

```
window.alert()
Math.PI
```



La page web affichée à l'écran est elle-même considérée par le JavaScript comme un ensemble d'objets imbriqués. Dans l'exemple ci-contre :

- L'objet fenêtre : ***window***
- L'objet document : ***document***
- L'objet formulaire : ***form***
- L'objet titre
- Etc.

Tout est objet !

B. L'objet *window*

L'objet *window* représente la fenêtre du navigateur. Il contient tous les autres objets. Ses très nombreuses propriétés et méthodes sont listées ici :
https://developer.mozilla.org/fr/docs/Web/API/Window

Voici 3 méthodes de l'objet *window* permettent d'afficher des fenêtres intruses (*pop-up*) :
`window.alert()` affiche une *pop-up* avec un texte d'alerte et un bouton "OK".
`window.confirm()` affiche une *pop-up* avec un texte et deux boutons : "OK" et "Annuler".
`window.prompt()` affiche une *pop-up* avec un champ texte à compléter.

Parce que l'objet ***window*** est **l'objet racine**, il est possible d'utiliser ses méthodes sans préciser le nom de cet objet. Ainsi, les deux écritures suivantes ont le même effet :

```
window.alert("Bonjour");
alert("Bonjour");
```

Exemple :

```
let age = prompt("Quel est votre âge ?");
if (age<18) alert("Ce site est interdit aux mineurs");
else let rep = confirm("Vous confirmez avoir plus de 18 ans ?");
```

C. L'objet *Math*

En JavaScript, ***Math*** est un objet natif permettant d'accéder aux fonctions mathématiques courantes : les fonctions trigonométriques, les arrondis, etc.

Méthodes :

```
Math.abs()          // valeur absolue
Math.sign()         // -1 si négatif ; 1 si positif ; 0 si nul
Math.ceil()         // borne supérieure
Math.floor()        // borne inférieure
Math.trunc()        // partie entière d'un nombre (sans les décimales)
Math.round()        // arrondi au plus proche

Math.pow(x, n)      // retourne x puissance n
Math.sqrt()         // racine carrée
Math.cbrt()         // racine cubique
Math.hypot(x, y)    // hypoténuse (racine carrée de la somme)
Math.cos()          // cosinus
Math.sin()          // sinus
Math.log10()        // logarithme en base 10


Math.min(4, 2, 8) // minimum (dans ce cas, retourne 2)
Math.max(4, 2, 8) // maximum (dans ce cas, retourne 8)

Math.random()       /* génère un nombre réel aléatoire entre 0 et 1
                    (valeur 1 non comprise). Exemple (entre 10 et 20) :
                    let x = Math.floor( Math.random()*11 + 10 ) ; */
…
```

Propriétés constantes :

```
Math.E              // constante e
Math.PI             // constante pi
Math.SQRT2          // constante racine de 2
…
```

D. *null* et *undefined*

Le mot ***null*** représente un objet vide, sans valeur.
Si l'objet existe, ses propriétés inexistantes ou sans valeur valent ***undefined***.

E. Le mot-clé *this*

Le mot-clé *this* fait référence à l'objet courant. Il s'agit d'un raccourci d'écriture bien pratique.

Par exemple lors d'un appel de fonction à partir d'un élément HTML, *this* fait référence à cet élément :

```
<div onclick="clic_div(this)">…</div>
```

Dans cet exemple, la fonction *clic_div()* recevra l'objet * * lors de son appel. Comprenez par « objet * * », l'entièreté du y compris ses attributs HTML, ses propriétés CSS, son contenu HTML, etc.

Attention aux fonctions fléchées qui présentent une différence avec les fonctions classiques dans le cas de l'utilisation de *this* : elles conservent le contexte d'appel de la fonction au lieu de l'objet courant.

F. Créer un objet

La syntaxe suivante, utilisant l'opérateur ***new***, crée une instance d'un certain objet à partir du constructeur (natif ou pas) prévu à cet effet (ici *Object()*). Une fois l'objet ainsi créé, on peut lui initialiser des propriétés.

```
let monSyllabus = new Object();
monSyllabus.titre = "Développement front-end";
monSyllabus.nbPages = 836;
```

Autres syntaxes :

```
let film = { titre:'Willow',  réalisateur:"Ron Howard",  annee:1988 };

let maVoiture = {
   marque: "Toyota",
   modele: "Yaris",
   couleur:'#ccc',
   annee: 1998
 };
```

Autre syntaxe, passant par la définition d'un constructeur :

```
// Définition de l'objet Bateau via un constructeur
function Bateau (nom, equipage, bordee, commandant) {
    this.nom = nom;     //this fait référence à l'objet Bateau
    this.equipage = equipage;
    this.bordee = bordee;
    this.commandant = commandant;
}

// Création d'une variable contenant une instance de l'objet Bateau
let bateau1 = new Bateau('Tonnerre de Grimnir', 3, 2, 'Brokk');

// Utilisation de l'objet
console.log('Quel navire ce ' + bateau1.nom + ' !');

// Les propriétés inexistantes d'un objet valent undefined
console.log(bateau1.voilure);      //affiche undefined dans la console
```

G. Supprimer un objet

L'opérateur *delete* sert à supprimer un objet ou une propriété d'un objet.

```
delete unObjet ;
delete unAutreObjet.unePropriete ;
```

H. Références

En JavaScript, les objets sont des références, contrairement aux variables. Ceci implique la différence de comportement suivante :

```
//Soit « x » un objet possédant une propriété « a »
let test = x ;
test.a = 88 ;     //cette instruction affecte « x.a »

//Soit « v » une variable
let test = v ;
test = 88 ;       //cette instruction n'affecte pas « v »
```

Les propriétés, comme les variables, ne sont pas des références :

```
// Soit « x » un objet possédant une propriété « a »
let test = x.a ;
test = 88 ; //cette instruction n'affecte pas « x »
```

Les tableaux étant des objets, ils présentent donc cette même particularité : affecter tout un tableau à une variable en fera une référence au tableau.

Cette particularité des objets s'applique également lors du passage d'arguments à une fonction :

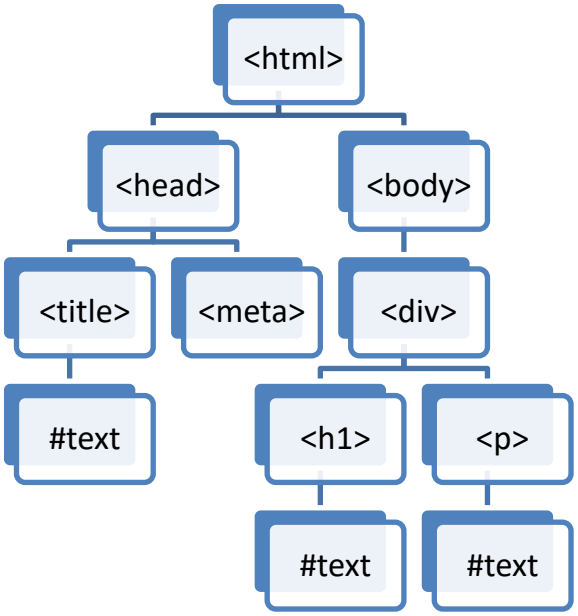
```
function incremente(tab, x) {
      tab[0]++;
      x++;
}

let T = new Array(5,10,20);
let A=5;
incremente(T,A);
console.log(T[0]);      //affiche 6
console.log(A);         //affiche 5
```

1.10 DOM

A. Introduction au DOM

Le **Document Object Model** (**DOM**) est une Interface de Programmation (API) pour les langages XML et HTML. Les API sont des ensembles d'outils permettant de faire communiquer entre eux des programmes ou des langages.
DOM est une technologie détenue par le W3C et qui en est actuellement à sa 3e version : DOM-3.

DOM considère les pages HTML comme des **arborescences où les éléments HTML et les textes constituent les nœuds.** La plupart des fonctionnalités JavaScript dans les sites web aboutissent à une manipulation du DOM : la manipulation d'un texte ou d'un élément de la page.




```
<html>
<head>
      <meta charset="UTF-8">
      <title>DOM</title>
</head>
<body>
<div>
    <h1>truc</h1>
    <p>machin</p>
</div>
</body>
</html>
```

B. Saturation du Thread principal

Attention, un DOM surchargé est un facteur de saturation du « thread principal ». Ce thread principal, c'est le processus chargé de la lecture du HTML, de sa mise en forme via les CSS, de l'analyse et de l'exécution du JavaScript, puis de la gestion des évènements, etc.

Saturer le thread principal provoque un délai avant l'affichage de la page. L'outil Page Speed Insight de Google fournit des informations supplémentaires et de précieux conseils sur ce sujet.

C. Accès aux éléments

L'objet ***document*** est l'objet racine du DOM : il donne accès à toutes les méthodes de recherche d'éléments de la page.

Accès par recherche

La méthode la plus classique est la recherche d'élément par ID :

```
document.getElementById('x')      // le premier élément d'ID x
```

Exemple :

```
<p id="alpha">Bonjour les gars !</p>
<script>
  const objet = document.getElementById('alpha');
  console.log(objet);   // Affiche « [objectHTMLDivElement] »
</script>
```

Les méthodes suivantes retournent plusieurs éléments. Remarquez le « s » à « Elements ». Leurs résultats sont des tableaux d'éléments particuliers appelés « HTMLCollection » et ne supportant par la méthode *forEach* sur certains navigateurs.

```
document.getElementsByTagName('p')      // les éléments <p>
document.getElementsByClassName('c')    // les éléments de class « c »
document.getElementsByClassName('c d')  // de class « c » et « d »
document.getElementsByName('nom')       // les éléments de name « nom »
```

Remarque : l'attribut *name* est présent dans les champs de formulaires.

Exemple :

```
<p class="brol">Coucou</p>
<p class="brol">Salut</p>
<script>
  const brols = document.getElementsByClassName('brol');
  for (b of brols) { console.log(b); }
</script>
```

Appliquées à l'objet *document*, ces méthodes de sélection de nœuds recherchent des résultats parmi tous les éléments de la page, depuis la racine. Mais il est possible de les appliquer à des nœuds pour restreindre la sélection. Cet exemple sélectionne les éléments de classe « c » contenus dans l'élément d'ID « x » :

```
document.getElementById('x').getElementsByClassName('c')
```

Accès par sélecteur CSS

La recherche d'éléments peut être faite par des sélecteurs CSS.

Retourner le premier élément du document correspondant au sélecteur CSS passé :

```
document.querySelector('#contenu .images')
```

Retourner tous les éléments du document correspondant au sélecteur passé :

```
document.querySelectorAll('#contenu .images')
```

Vérifier si un élément correspond à une sélection CSS :

```
x.matches('p.truc')   // renvoie true ou false
```

Accès par parentalité

Sélection d'éléments selon la parentalité :

```
x.children            // les éléments enfants de x
x.firstElementChild   // premier enfant de x
x.lastElementChild    // dernier enfant de x
x.parentElement       // parent de x
x.nextElementSibling      // prochain frère
x.previousElementSibling  // frère précédent
```

Accès direct

Il est possible d'accéder directement à certains éléments du DOM.

```
document.body         // le body
document.forms        // tableau contenant tous les formulaires
document.script       // tableau contenant tous les scripts
```

Enfin, bien que la syntaxe soit déconseillée, les éléments HTML possédant un ID sont accessibles par des variables globales du même nom.

```
<img id="loulou" src="loulou.png" alt="loulou">
<script>
  console.dir(loulou) ;
</script>
```

D. Modifier le contenu des éléments

La propriété ***innerHTML*** reprend tout le contenu d'un élément HTML : aussi bien le texte que les éléments enfants.

Les propriété ***innerText*** et ***textContent*** reprennent le contenu textuel d'un élément, si ce n'est que ***innerText*** ne retourne rien dans le cas d'un élément en *display:none* ou dans le cas des éléments *<script>* ou *<style>*.

Exemple : le code ci-dessous modifie le message « Coucou les enfants ! » en « Bonjour les gars ! » :

```
<div id="alpha">
  <p> Coucou les enfants ! </p>
</div>
<script>
  let element = document.getElementById('alpha');
  element.innerHTML = '<p><strong>Bonjour les gars !</strong></p>' ;
</script>
```

Les méthodes ***prepend()*** et ***append()*** permettent d'ajouter du contenu textuel en début ou en fin d'élément :

```
element.prepend('Vous me semblez très prometteurs…' );
element.append('Je suis fier de vous !' );
```

Supprimer un élément :

```
element.remove() ;
```

E. Propriétés des nœuds

Les nœuds peuvent être des éléments ou des textes.

`.childNodes`	nœuds enfants
`.firstChild`	premier nœud enfant
`.lastChild`	dernier nœud enfant
`.parentNode`	nœud parent
`.previousSibling`	nœud précédent d'un type (nœud de même niveau)
`.nextSibling`	prochain nœud d'un type (nœud de même niveau)
`.nodeName`	nom du nœud
`.nodeValue`	contenu du nœud
`.nodeType`	type du nœud

F. Propriétés HTML et CSS

En JavaScript, chaque élément HTML de la page est accessible sous forme d'objet. Ces objets disposent de leurs attributs HTML sous forme de propriétés. Par exemple, pour un objet *x* contenant un élément HTML *<img>*, *x* dispose des propriétés suivantes :

- `x.src`
- `x.alt`
- `x.width`
- `x.className // pour l'attribut « class »`
- …

La manipulation des attributs HTML peut aussi se faire avec les méthodes ***getAttribute()*** et ***setAttribute()***. Exemple affectant la source de l'élément *y* à l'élément *x* :

```
x.setAttribute('src', y.getAttribute('src') ) ;
```

De plus, ce même objet *x* dispose de l'objet ***style*** reprenant tous ses **styles CSS internes**. Les styles CSS provenant des feuilles de style ne sont donc pas concernés, seulement les styles codés dans l'attribut *style*. Le JavaScript ne supportant pas les tirets dans les noms, l'écriture des propriétés CSS est en ***camelCase*** : « fontSize » au lieu de « font-size ».

- `x.style.border`
- `x.style.left`
- `x.style.backgroundColor`
- `x.style.marginTop`
- …

Modifier le CSS d'un élément en passant par l'objet *style* ne modifie pas la feuille de style CSS, mais seulement le style interne de l'élément, comme dans l'écriture suivante :

```
<p style="margin-top:1rem;"> … </p>
```

G. L'attribut *class*

L'attribut HTML *class* peut contenir plusieurs valeurs séparées par des espaces. Dès lors, la modification de cet attribut est plus sensible que d'autres et requière l'utilisation des méthodes *add(), remove()* ou *toggle()* du sous-objet *classList*.

```
<script>
  const item = document.getElementById('alpha');
  item.classList.add('nomClasse') ;      //ajoute une classe
  item.classList.remove('nomClasse') ;   //enlève une classe
  item.classList.replace('old', 'new') ; //remplace une classe
  item.classList.toggle('nomClasse') ;   //alterne une classe
  item.classList.contains('nomClasse') ; //teste si possède une classe
</script>
```

1.11 Évènements

A. Les évènements

Les évènements sont des actions détectables de l'utilisateur : cliquer, survoler, charger la page, redimensionner la fenêtre, défiler la page, modifier un champ de formulaire, etc.

Évènements JavaScript	Attributs HTML	Descriptions
click	onclick	Le clic
dblclick	ondblclick	Le double-clic
mouseover	onmouseover	Survol
mouseout	onmouseout	Fin du survol
mouseenter	onmouseenter	Survol sans propagation
mouseleave	onmouseleave	Fin du survol sans propagation
mousedown	onmousedown	L'enfoncement du bouton de la souris
mouseup	onmouseup	Le relâchement du bouton de la souris
mousemove	onmousemove	Le mouvement du curseur sur un élément
pointerdown	onpointerdown	Extension des évènements « *mouse* » et fusion avec les évènements « *touch* » consacrés aux mobiles mais incompatibles sur Safari, les évènements « *pointer* » se préoccupent de la souris, du stylo ou stylet et des points de contact sur écran tactile.
pointerup	onpointerup	
pointerover	onpointerover	
pointerout	onpointerout	
pointerenter	onpointerenter	
pointerleave	onpointerleave	
pointermove	onpointermove	
load	onload	Le chargement d'une ressource et de ses ressources dépendantes ; souvent utilisé sur *body*
unload	onunload	La fermeture de la page
scroll	onscroll	L'utilisation des barres de défilement
resize	onresize	Le redimensionnement de la page
focus	onfocus	L'activation d'un champ du formulaire. L'évènement *focusin* est semblable, mais avec propagation.
blur	onblur	La désactivation d'un champ de formulaire. L'évènement *focusout* est semblable, mais avec propagation.
change	onchange	Lors de la désactivation, si changement de la *value* d'un champ *text* ou apparenté, ou de la sélection d'une option ou d'un fichier, ou le changement de l'état *checked* d'un bouton *checkbox*, ou le cochage d'un bouton *radio*.
input	oninput	L'entrée d'un caractère dans un champ texte
select	onselect	La sélection d'un texte dans un champ
submit	onsubmit	L'envoi d'un formulaire (uniquement sur <form>)
keydown	onkeydown	Touche du clavier enfoncée
keyup	onkeyup	Touche du clavier relâchée

Remarque : sous forme d'attribut HTML, le nom de l'évènement est précédé de « on » et n'est pas sensible à la casse.

De nombreux autres évènements existent en JavaScript et sont décrits dans la documentation officielle : https://developer.mozilla.org/fr/docs/Web/JavaScript

B. Écouter les évènements en JS

Le JavaScript est un **langage évènementiel** : il améliore l'interactivité des pages web en détectant les évènements et en les associant à des traitements grâce à la méthode *addEventListener()*.

On lance l'écoute du clic sur *x* (référence à un élément HTML) afin de déclencher une fonction :

```
x.addEventListener("click", maFonction) ;
```

Si la fonction nécessite des paramètres, on doit passer une fonction qui contient l'appel :

```
x.addEventListener("click", function () { maFonction(x, y) ; } ) ;
```

On peut aussi coder toute une fonction dans l'écoute d'évènement :

```
x.addEventListener("click", function maFonction() {
      …
} ) ;
```

Nous pouvons aussi arrêter l'écoute d'un évènement :

```
x.removeEventListener("click", maFonction) ;
```

Exemple affichant le nombre de clics sur un élément

```
<h1 id="moi">Cliquez-moi</h1>

<script>
let nb=0 ;
document.getElementById('moi').addEventListener('click', function() {
      alert(++nb) ;
});
</script>
```

Attention, cette fonction ne peut pas être appliquée à un tableau ou à une collection d'éléments HTML. Il faut l'appliquer aux éléments un à un… Exemples dans des boucles :

```
<img class="dinos" src="dino1.png" alt="">
<img class="dinos" src="dino2.png" alt="">
<script>
const dinos = document.getElementsByClassName('dinos');
for (d of dinos) {
      d.addEventListener('mouseenter', uneFonction);
}
const tabDinos = document.querySelectorAll(".dino");
tabDinos.forEach(function(element) {
    element.addEventListener("click", uneFonction);
});
</script>
```

C. Écouter les évènements en HTML

Il est possible de lancer des écoutes d'évènements depuis le HTML, en les codant sous leur forme d'attribut HTML : « click » devient « onclick ». Cependant, cette syntaxe est moins appréciée que la précédente car elle mélange le JS au HTML.

Exemple affichant le nombre de clics sur un élément

```
<h1 onclick="compte()">Cliquez-moi</h1>

<script>
let nb = 0 ;
function compte() {
      alert(++nb) ;
}
</script>
```

Ces écoutes sous forme d'attributs peuvent contenir une série d'instructions JavaScript séparées par des points virgules.

Exemple affichant des messages au chargement et à la fermeture d'une page web

```
window.addEventListener('load', function() { alert('Bonjour'); } );
window.addEventListener('unload', function() { alert('Bye bye'); } );
```

Version HTML

```
<body onload="alert('Bonjour');" onunload="alert('bye');"></body>
```

D. Le gestionnaire d'évènements

L'objet « **event** » est un gestionnaire d'évènements pouvant être passé en paramètre à la fonction appelée lors du déclenchement d'un évènement. Cet objet permet :

1. **De récupérer la position du curseur**

```
<img id="poule" src="galinette_cendree.png" alt="">

<script>
const poule = document.getElementById('poule');
poule.addEventListener('click', function(event) {
      console.log(`x:${event.clientX} y:${event.clientY}`) ;
});
</script>
```

Comme *clientX* et *clientY* sont les coordonnées du curseur par rapport à la page, *screenX* et *screenY* le sont par rapport à l'écran, et *offsetX* et *offsetY* par rapport au bloc.
Quant à *movement*X et *movementY* (à utiliser principalement avec l'évènement mousemove) ce sont les déplacements relatifs à la dernière position.

2. **De récupérer la touche du clavier qui a été pressée**

```
<script>
document.addEventListener('keydown', function(event) {
      console.log(event.key) ;
});
</script>
```

3. **De stopper la propagation d'un évènement**

   Lorsque l'on clique un élément HTML, l'évènement clic est déclenché sur cet élément, puis automatiquement sur son parent, sur le parent de son parent, et ainsi de suite jusqu'à la racine. C'est ce qu'on appelle la propagation.

   Il peut être utile de stopper cette propagation, lorsque des éléments imbriqués déclenchent diverses fonctions.

   `event.stopPropagation();`

4. **D'empêcher l'exécution de l'action par défaut**

   Empêcher un bouton de soumission d'envoyer un formulaire, empêcher un lien d'ouvrir une nouvelle URL, etc. C'est possible avec :

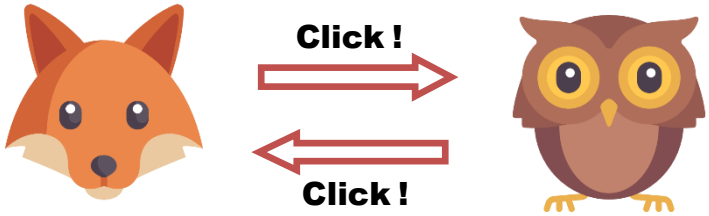
   `event.preventDefault();`

5. **De récupérer l'élément déclencheur ou l'élément surveillé**

```
<div id="dino"><img src="tyrannosaure.png" alt=""></div>

<script>
const dino = document.getElementById('dino');
dino.addEventListener('pointerover', function survol(event) {
      console.log(event.target) ;        // <img>
      console.log(event.currentTarget) ; // <div>
      console.log(this) ;                // <div>
});
</script>
```

   Différence subtile : dans le cas d'éléments imbriqués, *target* renvoie à l'élément ayant déclenché l'évènement, alors que *currentTarget* renvoie l'élément surveillant l'évènement.
   L'objet courant *this* revient ici à l'utilisation de *currentTarget*.

Exercice 1.1 : changer la source d'une image cliquée



Objectif
Développer une page HTML présentant une image dont la source change à chaque clic.

Étapes de réalisation

1. Choisissez 2 petites images.
2. Créez une page HTML comprenant un seul élément *<img>* affichant la 1re image. Attribuez à cet élément un id.
3. Codez un élément <script> et choisissez si vous allez coder le JavaScript dans cet élément ou dans un fichier externe.
4. En JavaScript, sélectionnez cet élément par son id avec *document.getElementById()* et associez lui l'écoute de l'évènement *click* avec *addEventListener()* afin d'appeler une fonction.
5. Cette propriété accède à la source de l'objet courant : this.src.
6. Afin de créer une alternance entre les 2 images à chaque clic, testez si la source de l'image se termine par le nom de la 2e image, pour lui donner comme source la 1re, ou sinon la 2e image. Quelle méthode des chaînes de caractères permet de vérifier si une chaîne se termine par ?

Exercice 1.2 : calcul du total

Objectif
Développez une page HTML reprenant un champ de type *number* demandant le nombre d'articles, un prix à l'unité et un total valant le produit de ces 2 valeurs.

Au chargement de la page ET lorsque le visiteur modifie le nombre d'articles commandés, le mot « article » prend un « s » si le nombre est supérieur à 1 et le total est recalculé.

Aide
La valeur d'un champ de formulaire correspond à son attribut ***value***.

Les textes de la page, eux, sont accessibles par la propriété ***innerText***.

Pour manipuler des données comme le total, le prix à l'unité ou le mot « article », il est conseillé de les placer dans des éléments ** ou *<b>* possédant un *id*.

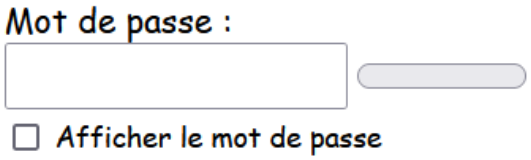
Il est souhaitable que le prix à l'unité soit chargé à partir de la page (et ne pas directement multiplier par 12.50€).

La fonction de calcul du prix et d'application du pluriel doit être appelée depuis 2 écoutes d'évènements : le changement de la valeur du champ ET le chargement de la page.

Exercice 1.3 : le mot de passe

Dans le petit formulaire suivant, nous retrouvons un champ de type *password*, une jauge <meter> et un champ de type *checkbox*.

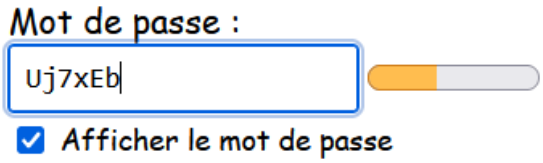
L'utilisation de <label>, toujours appréciée, permet d'étendre les zones cliquables des 2 champs.



Pendant que l'utilisateur entre son mot de passe, la valeur de la jauge (*value*) s'adapte, représentant ainsi la longueur du mot de passe.

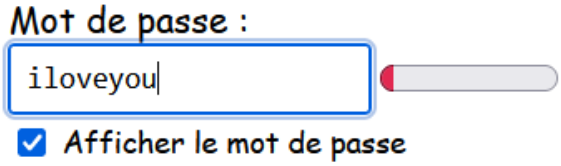
La jauge s'étalonne de 0 à 15 (*min* et *max*). Des paliers sont prévus à 5 et à 10 (*low* et *high*), de façon à ce que la jauge passe du rouge au jaune, puis au vert. Pour cela, la valeur optimale (*optimum*) doit être réglée sur 15.

Quand l'utilisateur coche la *checkbox*, le mot de passe devient visible. Pour cela vous aurez besoin de manipuler l'attribut *checked* de la *checkbox* mais aussi l'attribut *type* du champ mot de passe pour le passer de « password » à « text ».

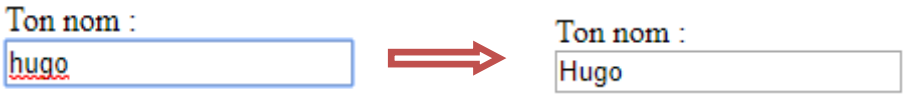


Enfin, une liste de mots de passe fortement déconseillés est utilisée pour signaler à l'utilisateur si son choix de mot de passe est trop courant. Pour cela, initialisez un tableau avec les valeurs suivantes : `"123456", "12345", "1234", "abc123", "abc", "pwd", "password", "mypassword", "dragon", "monkey", "shadow", "master", "superman", "spiderman", "batman", "god", "sex", "boobs", "azerty", "qwerty", "iloveyou", "computer", "welcome", "matrix", "secret", "login".`

Si le mot de passe est l'une de ces valeurs, la jauge passe à 1.



Exercice 1.4 : la majuscule



Objectif
Développez une page HTML contenant un champ texte où entrer son nom. Lorsque le visiteur sort du champ, transformez la première lettre du nom en majuscule.

Aide
Codez la partie HTML et donnez un ID au champ.

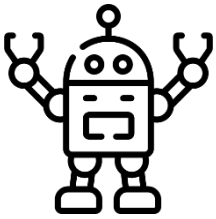
En JS, associez à ce champ l'écoute de l'évènement correspondant à la désactivation du champ et appelant une fonction.

À l'aide des méthodes des objets *string* (voir théorie), récupérez la première lettre du mot, transformez-la en majuscule, concaténez-la avec le reste du mot (sans la première lettre) et affectez le résultat à la valeur du champ.

N'hésitez pas à employer *console.log* pour consigner les valeurs sur lesquelles vous travaillez.

Testez si votre code fonctionne avec votre prénom mais aussi avec « émilie ».

Exercice 1.5 : déjouer les robots



La partie publique du Web est constamment sillonnée par des robots récupérant les adresses mails afin de constituer de gigantesques listings. Ces listings sont ensuite exploités à des fins publicitaires ou des arnaques en ligne.
Les développeurs souhaitent tromper ces robots patrouilleurs et font preuve d'imagination afin de leur dissimuler les adresses mails. Une méthode employée consiste à construire l'adresse avec du JavaScript, quelques fractions de seconde après le chargement de la page.

Voici à quoi ressemblent les emails de la page avant votre script :

```
<p>Adresses : <span class="email std">prénom.nom1</span>, <span class="email
std">prénom.nom2</span> et <span class="email">prénom.nomprof</span></p>
```

- ⇨ Codez un script surveillant le chargement de la page et de ses dépendances. Quelques centaines de millisecondes après ce chargement, appelez une fonction.

- ⇨ Cette fonction récupère tous les éléments de classe « email ». Pour chaque élément, s'il possède une classe « std », ajoutez « @std.heh.be » après l'élément, sinon ajoutez « @heh.be » après l'élément.

Exercice 1.6 : le formulaire

Objectif

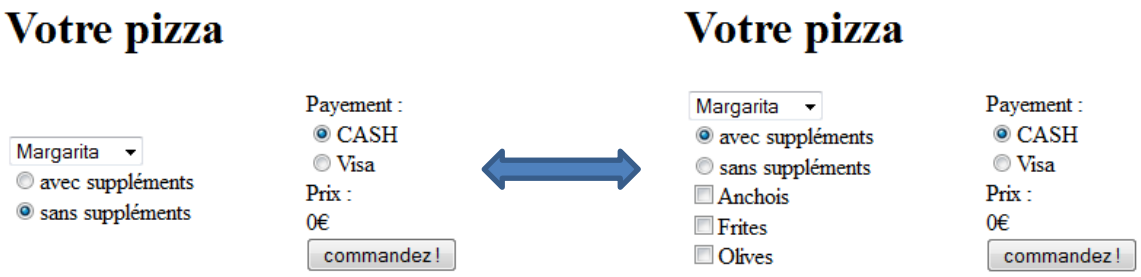
Développez une page HTML comportant un formulaire dont les suppléments sont visibles ou pas selon la case cochée et dont le prix est calculé selon une formule décrite ci-dessous.

1re étape

Les boutons radios « avec suppléments » et « sans suppléments » déclenchent l'apparition et la disparition des suppléments.

Le bouton « sans suppléments » est coché par défaut.

Il est conseillé de gérer cette apparition/disparition avec du CSS (modifié par JavaScript).



2e étape

Le prix doit être modifié via une fonction unique, à chaque clic sur des suppléments, sur des modes de payement ou sur « avec/sans suppléments ».

Le prix est égal au prix de la pizza (voir ci-dessous) + prix des suppléments (0.50€ par supplément) + 2€ si le mode de payement est Visa.

Afin de donner des prix différents aux différents choix de pizza, créez dans cette fonction, un tableau associatif dont les indices sont les value des <option> de la liste déroulante, et dont les valeurs sont les prix des pizzas ! Par exemple « Margherita » => 8€ ; « Regina » => 10€ ; « Capricciosa » => 11€ ; « Quattro Formaggi » => 10.50€ ; « Diavola » => 9.50€ ; « Marinara » =>12.50€ ; …

Exercice 1.7 (facultatif) : le Morpion

Principe
Le jeu du morpion comporte une grille de 3x3 cases et se joue à deux (deux couleurs de pions).

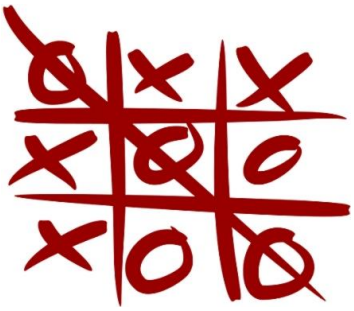
Les joueurs posent chacun à leur tour un pion dans l'une des cases. Le gagnant est celui qui arrive à aligner 3 pions de sa couleur.

Les joueurs ne peuvent pas déposer de pion dans des cases qui ne sont pas libres, ni changer leur pion de position une fois qu'ils sont déposés.

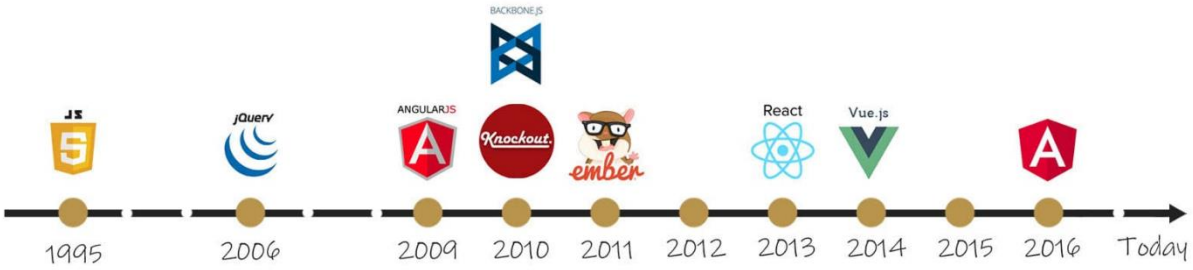
Le jeu s'arrête soit lorsqu'un joueur a gagné, soit lorsque la grille est pleine et que personne n'a gagné (égalité). Le résultat est annoncé aux joueurs par un message dans la page.

Fonctionnalités supplémentaires

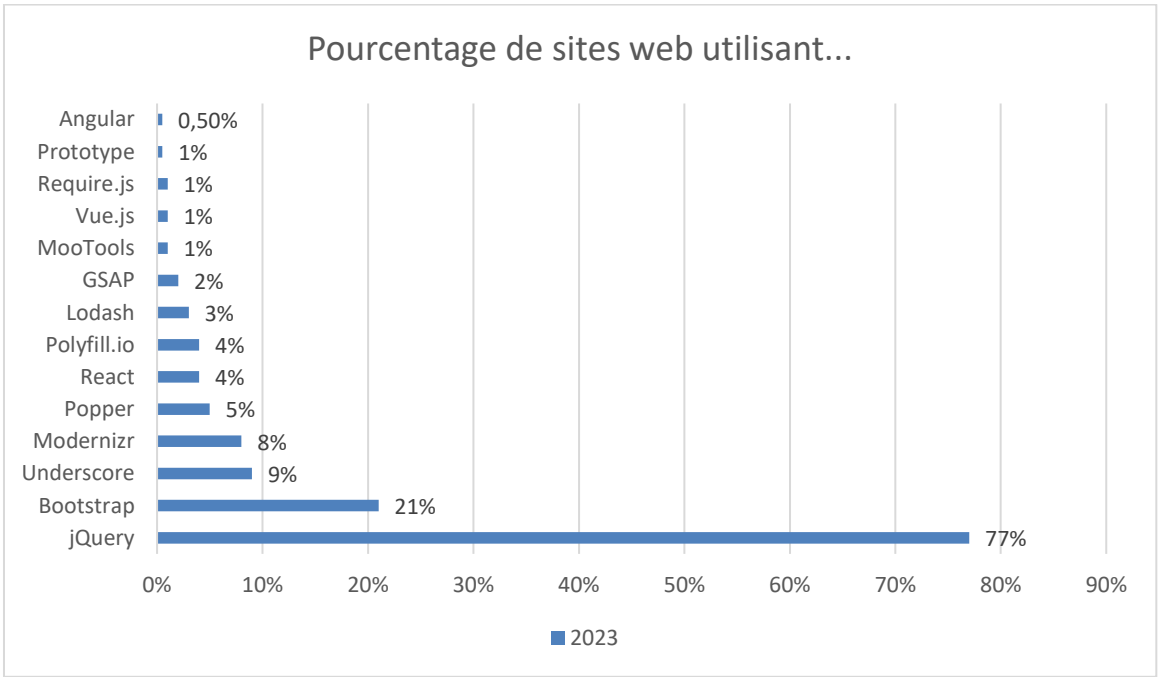
1. Proposez un élément visuel signalant la victoire d'un des joueurs.
2. Tirez au hasard le joueur qui débute la partie.
3. Créez un élément visuel montrant « à qui est le tour ».
4. Permettez aux joueurs de faire plusieurs parties de suite, et affichez le score de chaque joueur ainsi que le nombre d'égalités.
5. Permettez au joueur de choisir un mode de jeu : soit 1 contre 1, soit 1 contre l'ordinateur (bot).
6. Rendez votre bot le plus habile possible !



1.12 Les principaux frameworks JS



Les frameworks JavaScript (et outils apparentés) sont en plein essor depuis deux décennies. Parmi les plus célèbres, nous retrouvons React, Vue.js, Angular, Ember, Svelte, Bootstrap et jQuery.



Loin en tête de ce classement, jQuery reste le champion incontestable des frameworks JavaScript. Un doute subsiste tout de même autour de sa qualification de framework : de nombreux développeurs qualifient jQuery de bibliothèque. Ceci est dû à la légèreté de cet outil et au fait qu'il n'impacte pas l'architecture du projet, seulement la structure du code.

Le débat bibliothèque/framework concerne aussi React qui se définit comme une bibliothèque mais qui est souvent catégorisé comme framework… Force est de constater que certains outils se situent à la limite entre ces 2 catégories.

Définitions : framework et bibliothèque

Le **framework** est un ensemble de composants servant à la création de logiciels (ou de sites) et imposant une architecture ou des patrons de conception (*design patterns*) propres au framework.

La **bibliothèque** (*library*) est un composant n'imposant pas d'architecture ou de patron de conception à un projet informatique, mais apportant des fonctionnalités spécifiques.

Il est dès lors beaucoup plus aisé de changer de bibliothèque que de framework.

Le framework est donc générique (destiné à la création d'applications ou de sites**) alors que la bibliothèque est spécifique** (destinée à une fonctionnalité précise : les mails, le scroll, le son, etc.).

Exemples de **bibliothèques** JavaScript :

- Three.js pour la création de scènes 3D dans les pages web,
- D3.js pour la visualisation de données sous forme de tableaux, de diagrammes et de statistiques,
- wForms pour la gestion des formulaires,
- Des milliers d'autres…

Vanilla JS

Vanilla JS n'est pas un framework, c'est du JavaScript pur, sans bibliothèque.

Bootstrap

Développé au départ par Twitter, puis maintenu en projet open source sur GitHub, Bootstrap est un framework front-end, sous la forme d'une collection d'outils pour le développement de sites web. Ces outils sont pour la plupart des codes HTML et CSS, parfois JavaScript permettant l'ajout simplifié de modules (formulaires, icônes, boutons, éléments interactifs, etc.). Vu la grande communauté réunie par Bootstrap, il est aisé de trouver de l'aide en ligne ou des *templates* en téléchargement.

Vers 2010, alors que la mode des Smartphone se répand, Bootstrap apporte une solution à la problématique du Responsive Design et remporte un franc succès. Cette solution consiste à surcoder l'HTML en évitant de toucher au CSS, ce qui plait à bon nombre d'informaticiens mal à l'aise avec les feuilles de styles CSS…

Quelques années plus tard, les positionnements CSS3 *flexbox* et *grid-layout* solutionnent enfin la problématique de l'adaptativité des sites. La raison d'être de Bootstrap n'est plus. Pourtant le projet Bootstrap ne capote pas pour autant. Il faut dire que Bootstrap a eu le temps de se constituer une énorme communauté et de muter petit à petit en une boîte à outils de création de sites web très complète.

Foundation

Concurrent à Bootstrap, développé par ZURB, Foundation est un framework front-end gratuit et modulable.

Angular

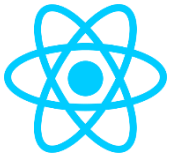
Développé par Google, Angular est un framework JavaScript open source côté client. Angular est spécialisé dans la création de SPA (*Single Page Application*) : des sites web mono-pages chargeant des parties de page plutôt que des pages entières, en vue d'améliorer l'expérience utilisateur.

Angular est qualifié d'arme lourde réservée à des projets web complexes.

Angular est basé sur une architecture de type MVC (Modèle Vue Contrôleur) : une structure de code qui simplifie la maintenance et le travail en équipe.

Angular recommande l'utilisation du langage TypeScript (sur-ensemble syntaxique de JavaScript).

React



React est une bibliothèque JavaScript libre développée par Meta (Facebook) depuis 2013. React (ou React.js ou ReactJS) a pour objectif de faciliter la création de SPA.

React ne gère que l'interface de l'application web (la vue dans un MVC) et complète volontiers des frameworks MVC comme Angular. L'écriture de cette vue fait intervenir un langage propre à React : le JSX.

React déploie un DOM virtuel ne mettant à jour le rendu de la page qu'en cas de nécessité.

Pour une utilisation optimale de React, Node.js est conseillé côté back-end.

Parmi ses utilisations les plus célèbres : Netflix, Yahoo, Sony, Facebook, Instagram, Whatsapp, etc.

Vue.js

Vue.js est un framework JavaScript *open source* permettant la création d'interfaces utilisateur et de SPA.

Svelte

Svelte est un framework JavaScript libre et à code ouvert écrit par Rich Harris en TypeScript.

Next.js

Next.js est un framework gratuit et open source s'appuyant sur la bibliothèque JavaScript React et sur Node.js.

jQuery

Mettons ce framework de côté : il vous sera présenté en détail au chapitre suivant !

A. Réflexion sur le coût des frameworks

La question des coûts engendrés par l'utilisation de frameworks JavaScript est de plus en plus posée. En effet : d'une part, les connexions internet sont de plus en plus rapides et les machines de plus en plus puissantes, alors que d'autre part, les performances des sites web semblent baisser. Parmi les principaux suspects de cette baisse de performance se trouvent les frameworks JavaScript.

Le JavaScript à outrance provoque, côté utilisateur, des surcoûts en téléchargement, en analyse des fichiers téléchargés, en exécution du code et en utilisation de la mémoire.

Alors que la plupart des sites web ne sont, au fond, que des pages d'informations textuelles sensées légères et performantes, la tendance actuelle est à l'exagération : toujours plus de code, plus de fonctionnalités, plus de cookies, plus de popups, plus d'images, plus de vidéos, plus d'animations, plus de traçage des visiteurs, etc.
Pour autant, ne freinons pas le progrès : c'est bel et bien l'utilisation intempestive des frameworks JS qu'il y a lieu de dénoncer et non pas l'évolution technologique du Web.

Plusieurs raisons peuvent expliquer l'utilisation abusive des frameworks :

- Des développeurs inexpérimentés appâtés par la mode des frameworks et par la promesse de coder plus facilement et plus rapidement.
- Des développeurs trop soucieux de leur productivité en dépit des performances du produit final. Rappelons que les performances des sites font partie intégrante du confort d'utilisation (ergonomie) et impactent le SEO.
- Une méconnaissance de l'HTML et du CSS, qui se substituent pourtant au JavaScript pour bon nombre de fonctionnalités de base tout en offrant de meilleures performances.

B. Expertise

Développeur expert en performances de sites web, Tim Kadlec, apporte des réponses pertinentes à cette question du coût des frameworks JS. D'après lui, il n'y a pas de moyen plus efficace de ralentir un site que d'utiliser un tas de JavaScript… et par « tas de JavaScript », il faut évidemment comprendre « framework JavaScript ».

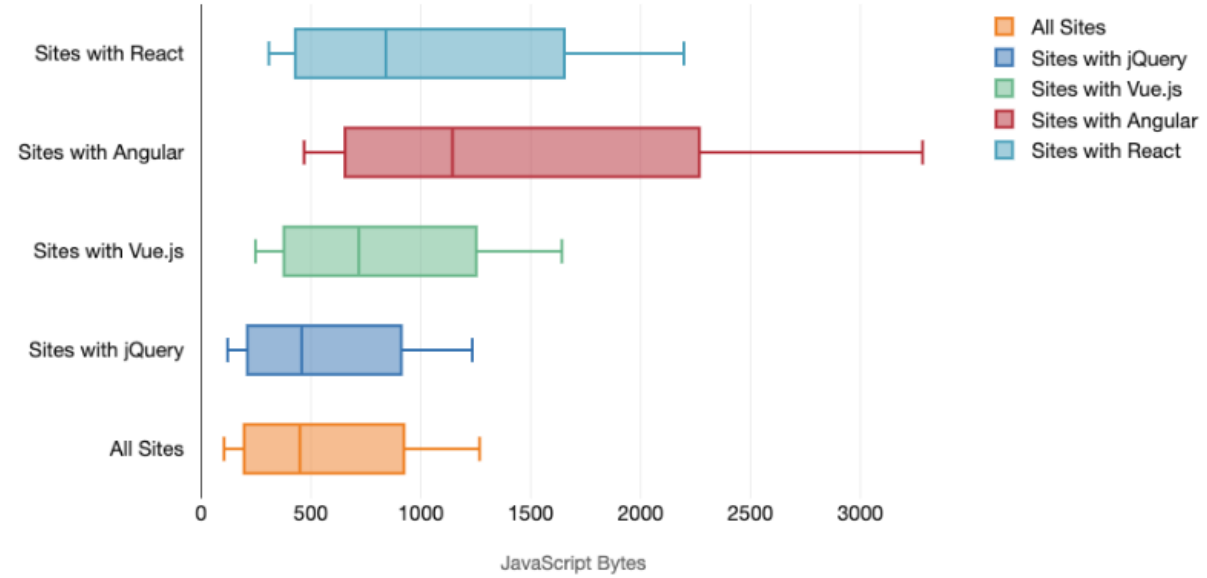
L'outil d'analyse « HTTP Archive » suit 4 à 5 millions d'URL, desquelles Tim en a déduit la fréquence d'utilisation des principaux frameworks JS :

Framework	*URL mobile*	*URL desktop*
jQuery	4 615 474	3 714 643
React	489 827	241 023
Vue.js	85 649	43 691
Angular	19 423	18 088

L'expert se penche ensuite sur plusieurs facteurs.

La quantité de code JavaScript téléchargée :

Quantité de JavaScript servi sur les appareils mobiles



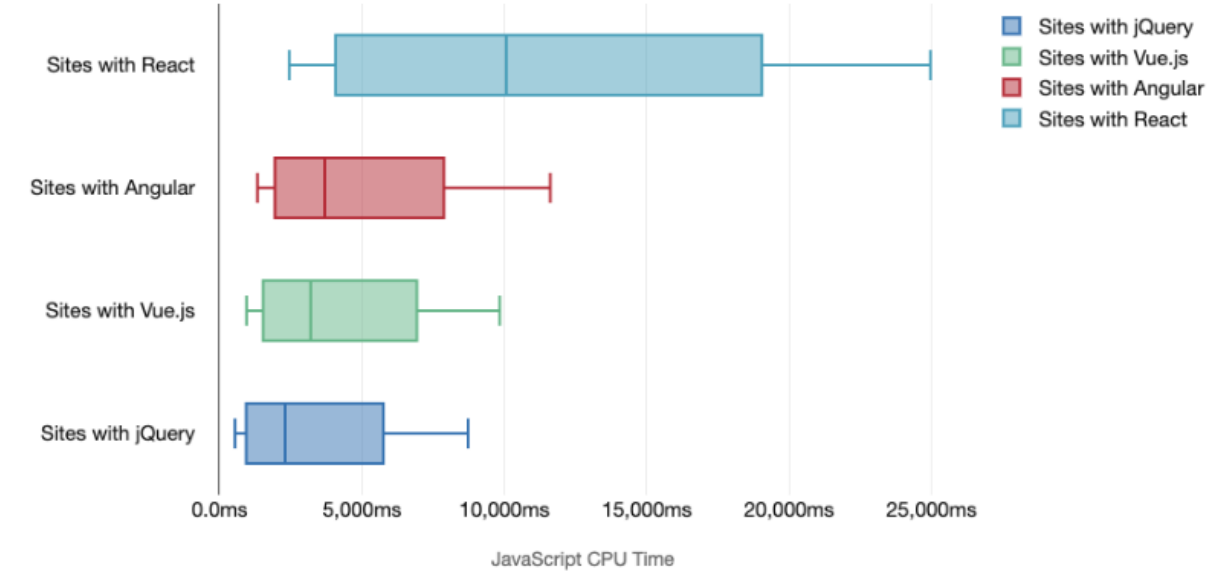
Quantité de JavaScript servi sur les appareils desktop

Le temps de travail du thread principal :

Pour cette étude, les sites comportant plusieurs frameworks ont été écartés.

Après le téléchargement du JavaScript, celui-ci provoque un travail non négligeable sur le thread principal du navigateur. Ce thread est chargé entre autres des entrées utilisateur et du rendu du site.

Temps de traitement CPU (en millisecondes) des scripts pour les appareils desktop



Temps de traitement CPU des scripts (en millisecondes) pour les appareils mobiles où un seul des frameworks est détecté

L'auteur de cette étude reste prudent sur les conclusions. D'abord, il y a lieu de remarquer que jQuery a des résultats proches du JavaScript pur. Seuls Angular et React provoquent des pertes de performance significatives.

Quant à ces pertes de performances, cette étude ne permet pas de déterminer si l'utilisation de ces frameworks étaient justifiées ou pas… Il est probable que bon nombre de sites utilisant ces frameworks justifient cette perte de performances par des fonctionnalités plus évoluées.

Un autre point de vue pertinent est celui de *cheatmaster30*, journaliste expert en jeu vidéo. D'après lui, le problème de performances des frameworks JS n'est pas la clé. Le vrai problème serait l'attitude des développeurs qui placent leur satisfaction personnelle au-dessus de celles des utilisateurs. Il estime que de nombreux composants sont ajoutés pour faire gagner du temps aux développeurs sans tenir compte de la surcharge de JavaScript ajoutée au produit fini.

Sources :
https ://javascript.developpez.com/actu/303519/Quels-sont-les-couts-lies-a-l-utilisation-de-frameworks-JavaScript-pour-le-developpement-Web-Une-analyse-des-sites-utilisant-React-Vue-js-ou-Angular/
https ://web.developpez.com/actu/289435/Les-frameworks-Web-detruisent-ils-vraiment-les-performances-du-Web-ou-l-experience-utilisateur-Ils-placeraient-la-satisfaction-des-developpeurs-au-dessus-des-utilisateurs/

C. Conclusion

En conclusion, insistons sur les avantages et inconvénients de l'utilisation de frameworks.
Les frameworks sont des couches supplémentaires. Ces couches provoquent des coûts et des gains qu'il est bon de mettre en balance lors des choix technologiques inhérents à chaque projet de développement.

Avantages	*Inconvénients*
- Impose une structuration du code, une discipline facilitant le travail en équipe et les opérations de maintenance. - Permet un gain de temps de développement (nombreux outils évitant de réinventer la roue) donc un meilleur rendement.	- Perte de performances : temps de chargement plus élevé et exécution du code plus lente. - Nécessite un apprentissage supplémentaire. - Comporte des risques liés à la durée de vie des frameworks. - Limite les fonctionnalités complexes (framework = cadre = limites). - Nécessite des mises à jour de version de framework en plus des mises à jour classiques des sites.

Chapitre 2 : jQuery

2.1 Introduction à jQuery

Créée en 2006 par John Resig, la bibliothèque **jQuery** s'est imposée rapidement et est devenue un outil incontournable du développement front-end.

Plus qu'une **bibliothèque**, jQuery se définit comme un ***framework*** JavaScript. Effectivement, jQuery n'est pas qu'un simple outil spécifique à une fonctionnalité, mais un véritable *framework* dans la mesure où son utilisation est générique et impacte la structure du code.

Son succès, **jQuery** le doit à ses 3 principaux avantages : une syntaxe plus simple que JavaScript, une gestion des incompatibilités JavaScript entre navigateurs, et des animations d'éléments aisées et personnalisables.

Aujourd'hui pourtant, l'utilisation de jQuery est remise en question. Nombre de développeurs appellent à son remplacement par du **Vanilla JS**, c'est-à-dire du JavaScript pur.

Il faut bien dire que depuis ECMAScript6 et plus récemment depuis le décès d'Internet Explorer, les problèmes d'incompatibilités JavaScript des navigateurs ont presque disparu. De plus, l'évolution du CSS3 permet dorénavant d'appliquer des transitions et des animations, remplaçant en partie les animations codées jadis en jQuery. Sans oublier que, comme tout outil, jQuery est une surcouche de code induisant une baisse des performances.

Benchmark comparant une sélection DOM jQuery avec 2 méthodes JS en nombre d'opérations par seconde. Source MeasureThat.net

Alors pourquoi continuer d'utiliser jQuery ? Tout d'abord, jQuery reste l'outil JavaScript le plus célèbre et le plus répandu au monde. Un incontournable. Vous en rencontrerez dans 75% des sites. Alors que les frameworks JS populaires comme Angular, React ou Vue.js ne dépassent pas les 5%...

Sans oublier que jQuery est compatible, léger (moins de 100 ko), gratuit, *open-source* et que des légions de *plugins* sont à disposition en téléchargement, permettant de mettre rapidement en place des *sliders*, des effets *lightbox*, des carrousels, des avertissements dans les formulaires, des éditeurs de texte WYSIWYG, etc.

Pas encore convaincus ? Attendez de voir la suite…

2.2 Mise en place de jQuery

La bibliothèque jQuery est téléchargeable à l'adresse :
http ://jquery.com/

Le site propose plusieurs versions :

- La version *compressed* (minimalisée) très légère car « compressée » mais illisible
- La version un*compressed* moins légère mais lisible
- Une version *slim* plus légère car amputée de son AJAX et de ses effets d'animation

Le fichier *jquery.js* se place dans le dossier de votre site ou de votre application, généralement dans un sous-dossier appelé « js » avec les autres fichiers JavaScript. Le code HTML suivant est nécessaire pour lier la bibliothèque à votre page web :

```
<script src="js/jquery.min.js"></script>
```

Le message d'erreur (voir la console du navigateur) « *$ is not defined* » ou « *jQuery is not defined* » vous avertit que la bibliothèque n'est pas correctement mise en place.

Remarque : cette fonction *$()* a priorité sur la fameuse fonction *window.onload()* disponible en JavaScript. L'ordre dans lequel se déroule l'interprétation du code par un navigateur est donc le suivant :

1. Le navigateur commence à télécharger les dépendances du fichier html : images, css, etc.
2. Le code situé dans *$()* est exécuté dès que possible.
3. Lorsque tout est téléchargé, le code situé dans *window.onload()* est exécuté, en dernier.

2.3 Fonctionnement de base

jQuery propose une fonction unique : ***jQuery( )***, accessible également via son alias ***$( )***.

Si cette fonction reçoit en paramètre une fonction, celle-ci sera exécutée juste après le chargement du DOM par le navigateur :

```
$( function () {
      // … code à interpréter après le chargement du DOM
}) ;
```

Mais le rôle principal de cette fonction est de **sélectionner des éléments HTML**. Voici différentes façons de procéder, en passant en paramètre :

- Un sélecteur CSS

```
$('.header')
$('#votreID')
$('p')
```

- Un objet faisant référence à un élément HTML

```
$(x)
/* x étant un objet HTML préalablement chargé par exemple,
de cette façon : let x = document.getElementById('votreID') ;
ou encore de cette façon : let x = $('#votreID') ; */
```

- D'autres objets JavaScript

```
$(document)
$(window)
$(this)
```

Dans tous ces cas, la fonction ***$( )*** retourne les éléments HTML correspondants sous forme d'un **objet jQuery** pouvant être manipulé, modifié, animé, etc. via les nombreuses méthodes jQuery présentées dans la suite de ce cours.

Contextes de sélection

Par défaut, le sélecteur jQuery recherche les éléments dans tout le document. Mais il est possible de restreindre la portée du sélecteur en **contextualisant** la sélection. Pour cela, il suffit de passer en 2e paramètre un objet HTML ou un autre sélecteur. La recherche du 1er paramètre se fera exclusivement dans le 2e.

Exemple :

```
$('p', '#footer')      // seuls les <p> dans #footer
$('p', this)           // seuls les <p> dans l'élément courant
```

Méthodes de traversées du DOM

La méthode *parent()* accède au parent d'un élément :

```
$('h2').parent() ;
```

La méthode *parents()* accède aux ancêtres d'un élément :

```
$('h2').parents() ;
```

La méthode *closest()* récupère le plus proche parent d'un type donné :

```
$('h2').closest('div') ;
```

La méthode *closest()* récupère le plus proche enfant d'un type donné :

```
$('h2').find('div') ;
```

La méthode *children()* accède aux enfants d'un élément :

```
$('h2').children() ;
```

La méthode *siblings()* accède aux frères d'un élément :

```
$('h2').siblings() ;
```

Utilisation « tableau » des sélections

Cet objet jQuery permet également une utilisation semblable à celle des tableaux JavaScript :

```
$('.header')[0]        //le premier élément de classe « header »
$('.header').length    //le nombre d'éléments de classe « header »
```

Dans le cas où les éléments retournés doivent subir un traitement un par un, il est possible d'exploiter cet objet jQuery avec la méthode *.each()* :

```
$('img').each( function(index){
  // Une ou plusieurs instructions JavaScript
}) ;
```

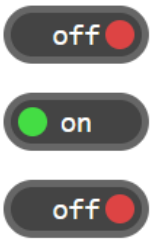
Remarque : dans cet exemple, la variable *index*, facultative, reprend le numéro de l'image (0 pour la 1re image, 1 pour la 2e, etc.)

Un exemple jQuery

Voici un exemple de 3 interrupteurs en HTML/CSS, dont l'interaction est gérée en jQuery.

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Exemple OnOff</title>
    <style>
      body { font-family:consolas,monospace ; }
      .onoff { margin:1rem ; width:64px ; height:16px ; background-
color:#444 ; border:4px solid #666 ; border-radius:16px ;
padding:4px ; cursor:pointer ; position:relative ; top:0 ; left:0 ; }
      .onoff :hover, .onoff :focus { background-color:#222 ; }
      .onoff p { position:absolute ; top:-1px ; left:0 ; right :0 ;
text-align :center; margin:4px ; color:#fff ; }
      .onoff div { width:64px ; background-color:#d44 ; width:16px ;
height:16px ; border-radius:8px ; position:relative ; top:0 ;
left:48px ; transition:400ms ;  }
      .onoff.on div { background-color:#4d4 ; left:0px ;  }
    </style>
  </head>
  <body>
    <div class="onoff"><p>off</p><div></div></div>
    <div class="onoff on"><p>on</p><div></div></div>
    <div class="onoff"><p>off</p><div></div></div>

    <script src="jquery.min.js"></script>
    <script>
      $( function() {
        $('.onoff').on('click', function() {
          $(this).toggleClass('on') ;
          if ( $(this).hasClass('on') ) $('p', this).text('on') ;
          else $('p', this).text('off') ;
        }) ;
      }) ;
    </script>
  </body>
</html>
```

Quand le bouton est « on ».



Une fois la page prête, le navigateur surveille le clic sur les boutons, afin d'alterner la classe « on » du bouton cliqué, et si le bouton cliqué possède la classe « on », il injecte le texte « on » dans le <p> du bouton, sinon il injecte le texte « off ».

2.4 Manipuler les éléments

jQuery permet de facilement modifier le document HTML (le DOM) : qu'il s'agisse des contenus textuels, des éléments HTML, de leurs attributs HTML ou de leurs propriétés CSS.

Certaines méthodes jQuery présentées ci-dessous s'utilisent de 2 façons :

- en ***getter*** afin de **récupérer** la valeur, pour cela on ne passe pas de paramètre
- en ***setter*** afin de **modifier** la valeur, pour cela on passe la nouvelle valeur en paramètre

Remarques : lors d'un ***getter***, si plusieurs valeurs sont trouvées, seule la première est retournée, sauf avec *text()* qui concatène les contenus textuels.
Le résultat obtenu peut valoir *undefined* si aucune valeur n'a été trouvée.

Accéder aux textes

Les méthodes suivantes manipulent uniquement le texte, pas les balises HTML.

Pour lire le contenu textuel d'un élément *(getter)* :

```
let titre = $('h1').text() ;
```

Pour remplacer le contenu textuel d'un élément par un nouveau contenu *(setter)* :

```
$('h1').text('nouveau texte') ;
```

Accéder au DOM

Les méthodes suivantes manipulent le DOM (balises HTML et textes).

Pour récupérer le DOM contenu dans un élément *(getter)* :

```
contenu = $('#exemple').html() ;
```

Pour remplacer le DOM contenu dans un élément *(setter)* :

```
$('#exemple').html('<p>Hop !</p>') ;
```

Pour insérer un extrait de DOM au début du contenu d'un élément :

```
$('#exemple').prepend('<p>Hop !</p>') ;
```

Pour insérer un extrait de DOM à la fin du contenu d'un élément :

```
$('#exemple').append('<p>Hop !</p>') ;
```

Pour insérer un extrait de DOM avant un élément :

```
$('#exemple').before('<p>Hop !</p>') ;
```

Pour insérer un extrait de DOM après un élément :

```
$('#exemple').after('<p>Hop !</p>') ;
```

Les méthodes suivantes sont semblables aux 4 précédentes mais ont une syntaxe qui place la destination à la fin de l'instruction plutôt qu'au début.

Insérer l'élément *#exemple* et son contenu, au début du contenu de *#destination* :

```
$('#exemple').prependTo('#destination') ;
```

Insérer l'élément *#exemple* et son contenu, à la fin du contenu *de #destination* :

```
$('#exemple').appendTo('#destination') ;
```

Insérer l'élément *#exemple* et son contenu, avant *#destination* :

```
$('#exemple').insertBefore('#destination') ;
```

Insérer l'élément *#exemple* et son contenu, après *#destination* :

```
$('#exemple').insertAfter('#destination') ;
```

La méthode *clone()* permet de cloner un élément afin de l'annexer ensuite au DOM avec par exemple l'une des méthodes précédentes.

```
$('#exemple').clone().insertBefore('#destination') ;
```

La méthode *replaceWith()* remplace les éléments sélectionnés par un nouveau code HTML :

```
$('hr').replaceWith('<br>') ;
```

La méthode *empty()* vide un élément de son contenu, alors que la méthode *remove()* supprime complètement un élément :

```
$('#exemple').empty() ;
$('#exemple').remove() ;
```

La méthode *wrap()* permet d'ajouter un parent à un élément. Ici, chaque <h2> sera enveloppé dans un :

```
$('h2').wrap('<div></div>') ;
```

La méthode *wrapAll()* permet d'ajouter un parent commun à plusieurs éléments. Ici, tous les <h2> seront enveloppés dans un seul global :

```
$('h2').wrapAll('<div></div>') ;
```

La méthode *wrapInner()* permet d'ajouter un parent au contenu d'un élément. Ici, chaque <h2> aura un enfant enveloppant son ancien contenu :

```
$('h2').wrapInner('<div></div>') ;
```

Accéder aux attributs HTML

```
largeur = $('#photo').attr('width') ;    // lecture
$('#photo').attr('width', '100px') ;     // modification(s)
$('.images').attr({src:'img.jpg', height:'100px', width:'100px'}) ;

$('img').removeAttr('width') ;           // supprime l'attribut width
```

Accéder aux valeurs des propriétés

La méthode **prop()** manipule l'état courant des attributs (après modification éventuelle), alors que la méthode **attr()** manipule l'état initial d'un attribut. C'est par exemple utile pour manipuler des champs de formulaires susceptibles de changer après le chargement de la page.

```
let ok = $('#bidule').prop('checked') ;  // lecture
$('option').prop('selected', False) ;    // modification
```

Accéder aux valeurs des champs de formulaire

```
$('#prenom').val('Bob') ;      // modifie la valeur du champ prénom
prenom = $('#prenom').val() ; // récupère la valeur du champ prénom
```

Accéder aux classes

```
$('#truc').addClass('red') ;       // ajoute la classe red
$('#truc').removeClass('red') ;    // enlève la classe red
$('#truc').toggleClass('red') ;    // alterne la classe red
$('#truc').toggleClass('red', condition) ;     // enlève la classe
      red si la condition est fausse, l'ajoute si la condition est
      vraie
```

Pour tester si des éléments possèdent une classe :

```
if ( $('#truc').hasClass('red') )
```

Pour tester si des éléments possèdent plusieurs classes :

```
if ( $('#truc').is('.big.red') )
```

Remarque : de manière plus générale, la méthode *is()* vérifie la correspondance entre au moins un des éléments de la sélection avec l'élément sélectionné par *is()*.

Accéder aux propriétés CSS

```
$('.photo').css('font-size') ;          //lecture
$('.photo').css('font-size', '20px') ;  //modification
```

Remarque : l'écriture 'fontSize' est également supportée mais toujours sous forme de chaîne de caractères.

Accéder aux dimensions

```
.width()      //largeur du contenu sans les marges (getter ou setter)
.innerWidth() //largeur padding compris (getter)
.outerWidth() //largeur padding et border compris (getter)
.outerWidth(true) //largeur padding, border et margin compris (getter)

.height()      //hauteur du contenu sans les marges (getter ou setter)
.innerHeight() //hauteur padding compris (getter)
.outerHeight() //hauteur padding et border compris (getter)
.outerHeight(true) //hauteur padding, border et margin compris (getter)
```

Accéder aux scrolls

```
.scrollLeft()     //nombre de pixels scrollés horizontalement
.scrollTop()      //nombre de pixels scrollés verticalement
```

Accéder aux positions

```
.offset() //position d'un élément dans la page (getter et setter)

.position() //position d'un élément dans son parent (getter)

$('#machin').offset().left //distance entre le bord gauche de la page et le bord gauche de l'élément

$('#machin').position().top //distance entre le bord supérieur du parent et le bord supérieur de l'élément
```

Ces méthodes ne renvoient que les écarts *left* et *top*, pas *right* et *bottom*.

Exercice écrit

Soit l'extrait HTML suivant :

```
<main>
 <section id="stock">
   <h2>En stock</h2>
   <article id="a473"> <h3>Article n°473</h3> <p>15€</p> </article>
   <article id="a270"> <h3>Article n°270</h3> <p>35€</p> </article>
   <article id="a168"> <h3>Article n°168</h3> <p>12€</p> </article>
 </section>
 <section id="rupture">
   <h2>En rupture de stock</h2>
   <article id="a119"> <h3>Article n°119</h3> <p>20€</p> </article>
 </section>
</main>
```

1. Proposez une instruction ajoutant l'attribut « id » de valeur « principal » à l'élément <main> :

2. Proposez une instruction modifiant le prix de l'article n°473 en 20€ :

3. Proposez une instruction ajoutant la classe « erreur » à l'article n°168 et à l'article n°119 :

4. Proposez une instruction appliquant une couleur rouge au texte des articles de la classe « erreur » :

5. Proposez une instruction ajoutant une <section id="avenir"> sous la section « rupture » :

6. Proposez une instruction injectant le titre de niveau 2 « À venir » dans cette nouvelle section :

2.5 Les évènements

A. Écouter les évènements

Les évènements sont les actions du visiteur : le clic, le survol, le *scroll*, …
La méthode jQuery *on()* permet de lancer l'écoute d'un évènement sur une sélection d'éléments. Elle prend en second paramètre une fonction à exécuter lors du déclenchement de l'évènement.

Lancer l'écoute de l'évènement *click* sur les images :

```
$('img').on('click', function() {
        //code exécuté à chaque clic d'une image
        $('img').removeClass('choix') ; // toutes les images
        $(this).addClass('choix') ; //seulement l'image cliquée
});
```

Remarque : dans la fonction, l'objet « this » fait uniquement référence à l'image cliquée, pas aux autres. L'objet « this » supporte les méthodes natives JavaScript, et supporte les méthodes jQuery s'il est passé à la fonction $: $(this).

Une erreur courante est de lancer une écoute d'évènement sur un élément qui n'aurait pas encore été chargé par le navigateur. Dans ce cas, l'écoute est perdue. Sauf si vous avez choisi de **déléguer l'écoute** :

```
$(document).on('click', 'img', function() {
        …
});
```

Remarque : si la page subit une modification du DOM, les images ajoutées seront concernées aussi par cette écoute d'évènement.

Au détour d'un script, vous rencontrerez peut-être cette ancienne syntaxe déconseillée :

```
$('img').click(function() { … }) ;
```

Arrêter l'écoute de l'évènement *click* sur les images :

```
$('img').off('click') ;
```

Déclencher l'évènement *click* depuis le code (sans intervention du visiteur) :

```
$('img').trigger('click') ;
```

B. Évènements

Consultez la liste d'évènements du chapitre JavaScript.

C. Gestionnaire d'évènements *event*

```
//Récupérer la touche appuyée
$(document).on('keydown',function(event) {
  console.log(event.key) ;
});
//Récupérer la position du curseur lors d'un clic
$(document).on('click',function(event) {
  console.log(event.pageX,' ',event.pageY) ;
});
```

D. Propagation d'évènements

Lorsque l'on clique un élément, le clic est détecté sur l'élément et sur chacun de ses ancêtres. C'est ce qu'on appelle la **propagation d'évènement**.
Imaginez un menu qui s'ouvre au clic d'un bouton et se ferme au clic n'importe où en dehors du menu. Si on surveille le clic sur le document, ce clic se déclenchera lors de n'importe quel clic, y compris le bouton d'ouverture du menu, ce qui aurait pour effet de l'ouvrir et de le fermer instantanément.
Pour corriger cela, il faut stopper la propagation :

```
$("#bouton-menu").on('click', function(event) {
      $("#menu").slideDown() ;
      event.stopPropagation() ;
});
$(document).on('click', function() {
      $("#menu").slideUp() ;
});
```

E. Chargement de la page

Il est de bon usage de commencer un script jQuery par attendre que le document soit prêt. Autrement, on risque de lancer des écoutes d'évènements ou des animations sur des éléments qui n'existent pas encore… ces écoutes ou animations seraient alors perdues.

```
$( function() {
  // Le document est chargé
});
```

Équivalent à :

```
$(document).ready( function() {
  // Le document est chargé
});
```

L'évènement « ready » n'existant pas en JavaScript, il n'est pas utilisable avec la méthode *on()*.

Il est également possible de s'inquiéter du chargement de la page dans sa totalité. Alors que *ready* s'assure que le parseur HTML du navigateur ait fini de construire le DOM, *$(window).on('load', …)* s'assure en plus que tous les fichiers (images, css, etc.) aient été chargés.

```
$(window).on('load', function() {
  // Le document et ses dépendances sont chargés
});
```

Exercice écrit

Soit l'extrait HTML suivant :

```
<article>
  <header>
    <h2>Titre de l'article</h2>
  </header>
  <div class="contenu">
    <p>Texte
  </div>
</article>
<div id="fleche"></div>
```

1. Proposez un code jQuery surveillant le clic sur l'élément <header> afin d'alterner la présence de la classe « ouvert » sur l'élément .

2. Soit l'élément positionné en absolu, proposez un extrait de code jQuery surveillant le survol de l'élément <article>, afin d'imposer à l'élément un écart *top* valant l'écart *top* de cet <article> par rapport à la page.

2.6 Animer les éléments

jQuery propose des méthodes d'animations à appliquer aux éléments HTML : *fadeIn() fadeOut(), fadeTo(), show(), hide(), slideDown(), slideUp(), slideToggle()* et *animate()*.

Durant ces animations, plusieurs propriétés CSS sont impactées, dont la propriété *display*. C'est pourquoi les effets d'apparition tels que *fadeIn(), show()* et *slideDown()* ne fonctionnent correctement que si les éléments sont masqués avec *display:none*.

```
$('.actu').fadeIn() ; // Apparition par variation d'opacité (1)
$('.actu').fadeOut(); // Disparition par variation d'opacité (0)
$('.actu').fadeTo(0.5); // Variation d'opacité
$('.actu').fadeToggle(); // Alterne entre fadeIn et fadeOut

$('.actu').hide(); // Cache instantanément ou progressivement
$('.actu').show(); // Affiche instantanément ou progressivement

$('.actu').slideUp(); // Apparition par variation d'hauteur
$('.actu').slideDown(); // Disparition par variation d'hauteur (0)
$('.actu').slideToggle(); // Alterne entre slideUp et slideDown

$('.actu').stop(); // Arrête l'animation en cours
$('.actu').stop(true); // Arrête les animations en cours et en attente
$('.actu').stop(true,true); // Idem et applique l'aspect final

$('.actu').delay(400); // Marque une pause
```

Durée et vitesse d'animation

Chaque méthode d'animation est personnalisable en termes de durée et de vitesse.
Voici 2 exemples équivalents présentant la durée par défaut et la vitesse par défaut :

```
$('.actu').fadeIn('normal', 'swing') ;
$('.actu').fadeIn({duration:'normal', easing:'swing'}) ;
```

La durée
La durée d'animation par défaut est *'normal'*, soit 400ms. Mais la vitesse peut également prendre comme valeur :

- 'slow', 'normal' ou 'fast' ; // 600ms, 400ms ou 200ms
- une valeur de type *number* comme 550 ou 8000 (interprétée en millisecondes) ;

Trois exemples d'instructions équivalentes :

```
$('.actu').fadeIn('slow') ;
$('.actu').fadeIn(600) ;
$('.actu').fadeIn({duration:600}) ;
```

Il est possible de redéfinir les durées par défaut et d'introduire vos propres durées personnalisées :

```
$.fx.speeds.fast = 300;
$.fx.speeds.bipbip = 50;
```

La vitesse
La vitesse d'animation par défaut est *'swing'*, soit une accélération au début suivie d'une décélération à la fin. L'autre valeur possible est *'linear'* pour une vitesse linéaire (constante). De nombreuses autres vitesses d'animations sont disponibles en ajoutant le plugin Easing.

Deux exemples d'instructions équivalentes :

```
$('.actu').fadeIn('linear') ;
$('.actu').fadeIn({easing:'linear'}) ;
```

Animation personnalisée avec *animate()*

Parmi toutes ces méthodes d'animation, ***animate()*** offre des possibilités de personnalisation d'animation plus poussées en permettant d'animer toutes les propriétés CSS dont la valeur est numérique : *height*, *width*, *padding*, *top*, *left*, *opacity*, *font-size*, etc. mais ne peut pas animer des propriétés prenant comme valeur une couleur, une URL ou des mots-clés.

Citez 5 propriétés CSS qui ne peuvent pas être animées avec *animate()* !

Exemple d'*animate()* :

```
$(this).stop().animate({height:'400px', borderWidth:'10px'});
```

Remarquez que les propriétés CSS dont le nom est composé s'écrivent en *camelCase*, suivant la syntaxe JavaScript. jQuery supporte aussi l'écriture CSS (mots séparés par des tirets) mais seulement sous forme de chaîne de caractères.
Voici 3 instructions équivalentes faisant varier la taille de la police :

```
$(this).stop().animate({fontSize:'80px'});
$(this).stop().animate({'fontSize':'80px'});
$(this).stop().animate({'font-size':'80px'});
```

Voici 2 autres exemples d'*animate()* personnalisés en durée et en vitesse :

```
$('.actu').stop().animate({fontSize:'80px'}, 'normal', 'swing');
$('.actu').stop().animate({fontSize:'80px'}, {duration:'normal',
easing:'swing'});
```

Comment choisir entre jQuery et CSS3 ?

Grâce aux nouvelles propriétés ***transition*** et ***animation***, CSS3 a peu à peu remplacé jQuery pour de nombreuses animations simples — et vu la facilité de syntaxe, c'est plutôt une bonne chose !

En matière d'animations, les animations vraiment complexes nécessitent l'utilisation de Canvas ou de vidéos. Pour des animations interactives comme des *sliders*, des carrousels ou des *lightbox*, l'usage de la bibliothèque jQuery reste très pertinent. En revanche, pour des effets plus simples, comme un changement de couleur ou de forme au survol ou au clic d'un élément, CSS3 suffit largement.

Ordre des animations

Par défaut, si 2 animations portent sur un même élément, elles ont lieu successivement. Si elles ont lieu sur des éléments différents, elles ont lieu simultanément.

Animations successives sur un même élément

Exemple avec *slide* :

```
$("#truc").slideUp().slideDown();
```

Autre Exemple avec *slide* :

```
$("#truc").slideUp();
$("#truc").slideDown();
```

Remarque : avec *animate()*, il est possible de faire varier plusieurs paramètres simultanément.

Animations simultanées sur des éléments différents

Exemple de 2 animations simultanées :

```
$("#truc1").slideUp();
$("#truc2").slideUp();
```

Animations successives sur des éléments différents

Exemple de 2 animations successives :

```
$('#truc1').slideUp({duration:"nomal",queue:true,
complete : function() {$("#truc2").slideUp("normal");}});
```

Exercice écrit

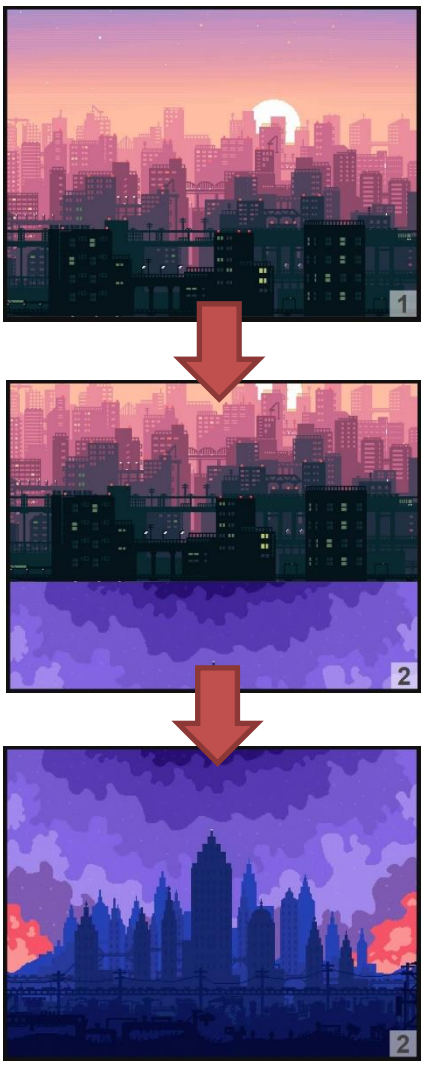
Soit l'extrait HTML suivant :

```
<div id="popup" style="display:none;">
  <h2></h2>
</div>
```

Proposez un extrait de code jQuery surveillant la préparation du document, puis animant avec un effet de fadeIn() rapide ; puis surveillant le clic sur cet élément pour lancer un effet de fadeOut() lent sur cet élément.

Exercice 2.1 : *Slider*

* Base HTML/CSS disponible sur Moodle *



Cet exercice consiste en un *slider* d'images réalisé en HTML/CSS et animé en jQuery.

Lors du clic sur le *slider*, il passe à l'image suivante en effectuant un mouvement vertical. Le numéro de l'image est affiché en bas à droite et se met à jour à chaque clic. Une fois la dernière image cliquée, le *slider* revient sur la première.

La structure HTML présente une lucarne de 800x600 dont ce qui dépasse est caché. La lucarne contient un plateau de dimension plus grande. Le plateau contient les images de 800x600 chacune. À la fin de l'exercice, une balise <img src="landscape.jpg" alt=""> sera ajoutée dans le plateau, pour vérifier si votre code est capable de s'adapter automatiquement.

1re étape (jQuery) : Vérifiez que le document est prêt, puis déclarez une variable *position=1* et une variable *nbImages*. Affectez à *nbImages* le nombre d'images dans le plateau récupéré dynamiquement avec jQuery.

2e étape (jQuery) : Lancez ensuite l'écoute du clic sur la lucarne, afin :

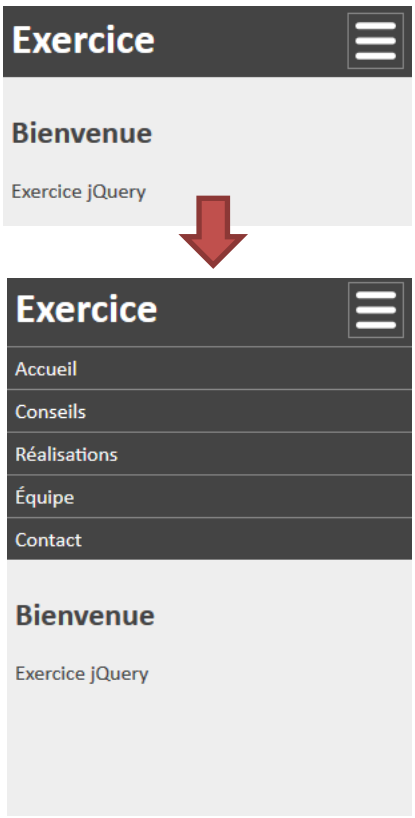
- D'incrémenter *position* à chaque clic
- Si la nouvelle valeur de position est supérieure à *nbImages*, la repasser à 1
- D'injecter la valeur de *position* dans l'élément #num

3e étape (jQuery) : Toujours au moment du clic de la lucarne, en jQuery, modifiez la propriété CSS *top* du plateau. Dans le CSS, ajoutez une transition au plateau.
Attention l'image 1 est en top:0px, l'image 2 est en top:-600px, l'image 3 en top:-1200px, etc. Il vous faudra trouver une formule utilisant la valeur de *position* pour calculer le bon *top*… sans oublier de concaténer l'unité « px » à la fin !

4e étape (HTML) : Ajoutez l'image dans le plateau en ne modifiant que le code HTML. Si vous avez bien travaillé, le *slider* s'adapte.

Exercice 2.2 : menu déployable pour mobile

* Base HTML/CSS disponible sur Moodle *



Dans cet exercice, nous allons compléter un code HTML/CSS afin d'animer le déploiement du menu avec jQuery.

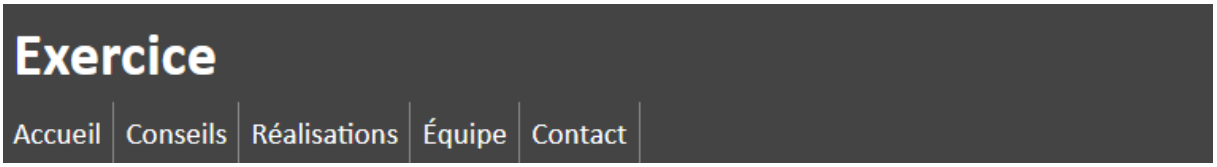
Certains sites optent pour un déploiement instantané. Ceci est facilement réalisable en modifiant la propriété CSS display. Pour notre part, nous allons animer ce déploiement avec une animation *slideToggle* en jQuery !

1re étape (CSS) : Dans le code, le menu « #navTop » est contenu dans « #volet ». Commençons par cacher le volet via CSS en lui appliquant un *display:none*.

2e étape (jQuery) : En jQuery, assurez-vous que le document soit prêt, puis détectez le clic sur #burger afin d'exécuter un *slideToggle()* sur *#volet*.
La méthode *slideToggle* se charge de faire passer le *display* de *none* à *block* et varier la hauteur du volet.

3e étape (CSS) : Les menus *burgers* sont utiles sur mobiles mais plutôt gênants sur *desktop*. En CSS, codez un media-query ne s'appliquant qu'aux largeurs de fenêtres supérieures à 800px afin de :

- Cacher l'icône du burger (le trigramme)
- Réajuster le menu pour un affichage à l'horizontal avec des bordures verticales du côté droit des liens.

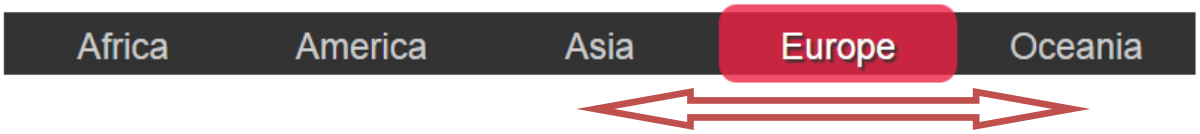


Exercice 2.3 : la bubulle

* Base HTML/CSS disponible sur Moodle *

Dans cet exercice jQuery, une navigation est ornementée d'une bulle qui met en évidence le lien survolé et qui disparait (fondu) lorsque l'on ne survole plus le menu.

La bulle ne suit pas exactement le curseur : elle n'effectue un déplacement qu'au moment où le curseur entre dans un nouveau lien.




```
<nav id="navbar">
      <div id="menu">
            <a href="#">Africa</a>
            <a href="#">America</a>
            <a href="#">Asia</a>
            <a href="#">Europe</a>
            <a href="#">Oceania</a>
      </div>
      <div id="bubulle"></div>
</nav>
```

1re étape : jetez un œil au code HTML/CSS

1) Quelle est la position de #navbar ?
2) Quelle est la position de #bubulle ?
3) Quelle est la position de #menu ?

➔ Pour cet exercice, vous devez régler la profondeur (*z-index*) de chaque élément : la bulle doit passer pardessus #navbar mais par-dessous #menu. Dans le cas contraire, la bulle empêcherait le survol des liens du #menu.

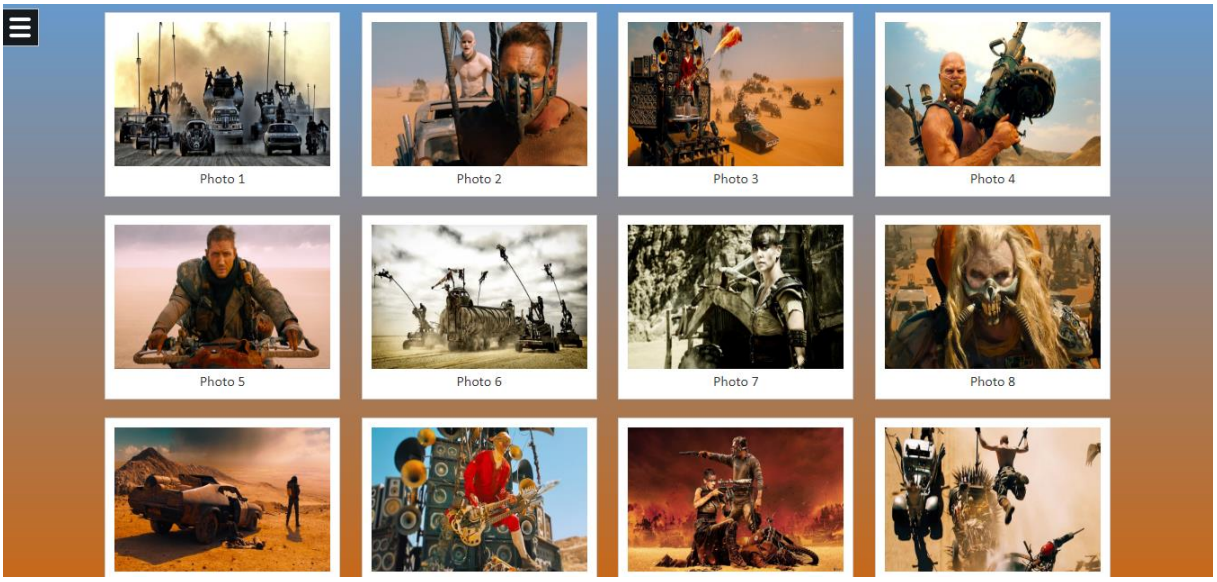
2e étape : animations CSS et jQuery

1) En CSS, faites apparaître ou disparaître la bulle en jouant sur son opacité, selon le survol de la barre de navigation.
2) En CSS, appliquez à tous les changements d'état de la bulle, dont cette apparition/disparition, une transition de 600ms en *desktop* et de 400ms en mobile.
3) En jQuery, détectez l'entrée du curseur sur les liens afin de modifier la propriété *left* de la bulle en lui passant l'écart *left* du lien survolé (*.position()* ou *.offset()* ?), et *width* en lui passant la largeur du lien survolé.
4) Agissez de même avec les propriétés *top* et *bottom* afin d'obtenir le même effet sur mobile, lorsque la navigation est présentée en colonne. Attention, la bulle est sensée dépasser du lien survolé de 8px en haut et en bas.

Exercice 2.4 : Galerie avec effet *lightbox* et volet de navigation

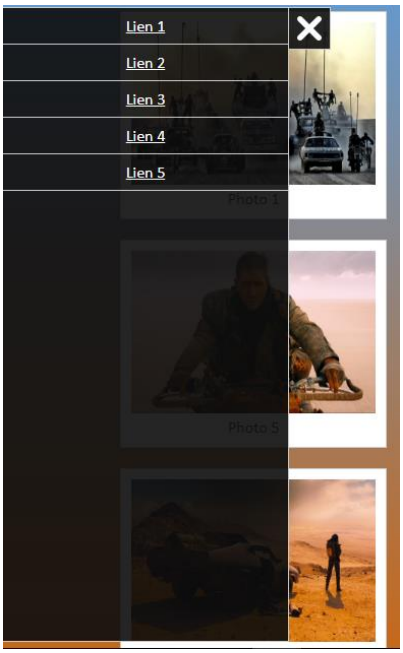
Nous allons développer une galerie de vignettes comportant 2 animations : un menu fixe déployable et un effet *lightbox* maison.

1re étape : Pour coder la galerie, utilisez les éléments <figure> <img> et <figcaption>. Choisissez de préférence des images de grande résolution (entre 1024px et 1920px de large). En CSS, le *flexbox* est le moyen le plus indiqué pour disposer les vignettes dans cette galerie.



2e étape : Chaque vignette (pas seulement l'image, toute la vignette) est cliquable afin de faire apparaître le *lightbox* : l'image en grand (sans dépasser et sans toucher les bords) sur un élément noir transparent recouvrant toute la page et ne défilant pas lors du *scroll*.

3e étape : Un clic n'importe où dans la surface du *lightbox* provoque la disparition du *lightbox* par diminution de l'opacité.
Donnez également la possibilité à l'utilisateur de fermer la *lightbox* en appuyant sur ESC.



4e étape : Le volet de navigation caché dans la marge ne laisse paraitre qu'un bouton avec une icône ☰. Ce symbole appelé « *trigram for heaven* » en anglais ou « trigramme du ciel », a adopté l'appellation « hamburger » lorsqu'il est utilisé dans le cadre de la navigation d'un site ou d'une application.

L'icône est appliquée en CSS avec *background-image* et non pas en HTML.

En jQuery, détectez si le hamburger est cliqué et utilisez *toggleClass()* pour alterner une classe (nom au choix) sur la navigation. Avec le CSS, utilisez cette classe pour décrire le comportement de la navigation déployée, puis utilisez *transition* pour animer le changement d'état.

Étape bonus : faites-en sorte que le volet puisse être fermé à partir d'un clic n'importe où dans le document. Attention, ceci nécessite de stopper la propagation du clic sur le burger.

Étape bonus : faites-en sorte que le clic sur la *lightbox* provoque un *fadeOut()* / *fadeIn()* sur l'image de la *lightbox*, avec un changement de source entre les deux de façon à passer d'une image à l'autre et fermeture qu'après la dernière image.

Exercice 2.5 : Lazyload et Scroll

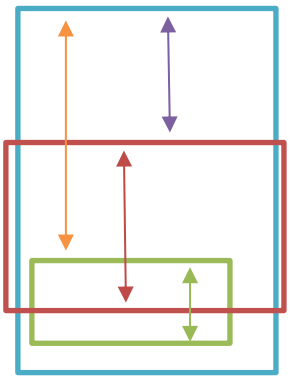
Dans cet exercice nous allons travailler sur le défilement de la page ou *scroll* en anglais, afin que des éléments apparaissent un peu après avoir franchi le seuil de scroll. Cette apparition différée est appelée « chargement paresseux » ou *lazyload* en anglais.

Pour cet exercice, vous aurez besoin d'un contenu suffisamment conséquent pour permettre le scroll vertical.

Lorsque le visiteur va scroller la page, il constatera avec plaisir qu'il déclenche de petites animations sur les contenus, dégageant ainsi l'impression que la page se construit petit à petit. Ces animations seront, pour cet exercice, des changements d'opacité à sens unique (une fois apparu, l'élément ne disparait plus).

1. En HTML, choisissez quelques éléments de la page qui vont subir cet effet d'apparition en leur codant la classe « apparition ».
2. En jQuery, une fois le document prêt, ajoutez aux éléments de classe « apparition », une classe « invisible ».
3. En CSS, appliquez une transition de 1s aux éléments de la classe « apparition ».
4. En CSS, sous l'instruction précédente, appliquez une transition de 0s et une opacité de 0 à la classe « invisible ».
5. En jQuery, écoutez l'évènement scroll et testez pour chaque élément de classe « invisible », s'il franchit le seuil de scroll (ou plutôt si la moitié de l'élément l'a franchi). Si c'est le cas, enlevez-lui la classe « invisible ».

Pour rappel, toute page web affichée par le navigateur est un objet ***window*** contenant un objet ***document*** contenant les **éléments HTML**. L'objet *window* possède les dimensions de la fenêtre du navigateur. L'objet *document* possède les dimensions du contenu HTML, et peut donc dépasser de *window*. C'est pourquoi l'objet *window* permet de faire défiler l'objet *document*, c'est ce qu'on appelle le *scroll*.



Pour détecter le moment où un élément entre dans la zone visible, en jQuery, surveillez l'évènement *scroll* sur l'élément *window*, ensuite utilisez les 4 valeurs suivantes pour tester si **l'élément** est à moitié dans la **zone visible** ou pas :

- **Le niveau de scroll de la fenêtre : $(window).scrollTop()**
- **La hauteur de la fenêtre : $(window).height()**
- **La hauteur de l'élément : $(…).outerHeight()**
- **L'écart *top* de l'élément par rapport au document : $(…).offset().top**

Exercice 2.6 : Parallaxe

La parallaxe est l'effet du changement de position de l'observateur sur ce qu'il perçoit. Dans une page web, lors du *scroll,* faire défiler les éléments à des vitesses différentes provoque une impression de profondeur.

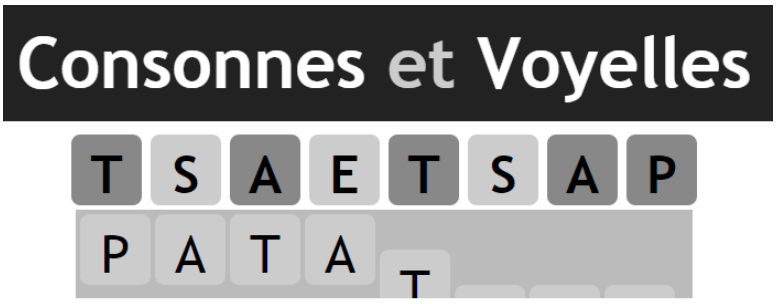
Dans cet exercice nous allons travailler sur le défilement de la page ou *scroll* en anglais, afin de développer un background semi-fixe donnant une impression de parallaxe.

Avec de bons yeux, vous constaterez que la série de *printscreens* ci-dessus contient un background qui n'est pas fixe, mais qui ne scrolle pas non plus à la même vitesse que la page.

Pour obtenir cet effet de parallaxe, commencez par appliquer une image en background à un élément de belle taille.

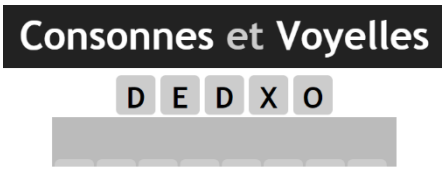
Ensuite, avec jQuery, détectez l'évènement *scroll* sur *window*, afin de modifier la valeur de la propriété CSS *background-position* de l'élément contenant le background, de façon à ce que la composante verticale dépende du niveau de scroll de la fenêtre : ***$(window).scrollTop();***

Exercice 2.7 : Consonnes et Voyelles



Objectifs

Au chargement de la page, les mots « Consonnes » et « Voyelles » du titre sont cliquables. Chaque clic génère une lettre aléatoire. Procédez en déclarant un tableau de consonnes et un tableau de voyelles parmi lesquels vous piochez des lettres.



La seconde étape survient une fois que les 8 lettres sont choisies (pas avant) : les 2 mots du haut ne produisent plus de lettre lorsqu'ils sont cliqués. Mais les 8 lettres deviennent alors cliquables et chaque clic, fait apparaître la lettre cliquée avec une animation jQuery : le bloc se déplace verticalement vers le haut.

Une fois une lettre cliquée, elle change de couleur et les futurs clics sur cette même lettre n'auront aucun effet.

Exercice 2.8 : Tilapins

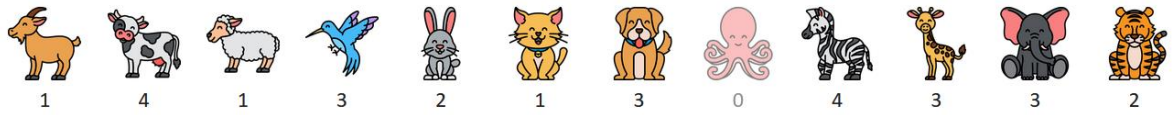
Fichiers nécessaires disponibles sur Moodle.

31

Au chargement, la page comporte :

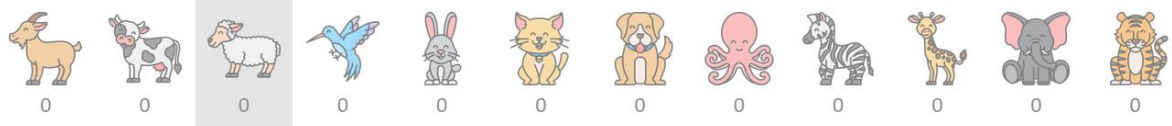
- Un *header* contenant 12 boutons, chacun contenant une image d'animal et un compteur
- Un 2e *header* contenant le total des compteurs
- Un *main* vide

Au chargement de la page,
    Pour chaque animal,
        Le compteur prend une valeur aléatoire entière comprise entre 1 et 4.
    Le total prend pour valeur la somme des compteurs.



27

Au clic sur un animal,
    Si le compteur de l'animal est supérieur à 0,
        Une image de l'animal cliqué est ajoutée à la fin du *main*
        Le compteur sous l'animal cliqué diminue de 1
        Le total s'adapte
    Si le compteur de l'animal passe à zéro,
        L'animal et son compteur deviennent transparents en recevant la classe "vide"
    Si le total passe à zéro,
        Le total devient vert (classe "vide")



0

Chapitre 3 : AJAX

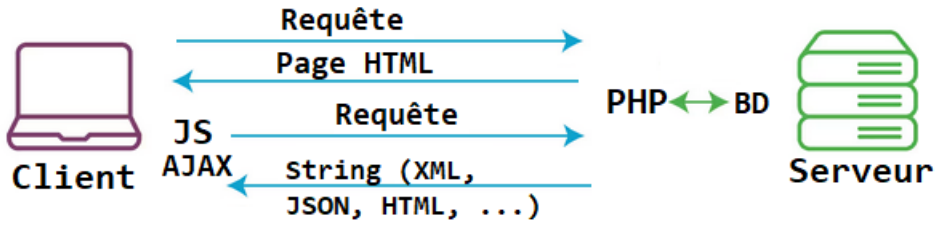
3.1 Introduction à l'AJAX

AJAX, ou *Asynchronous JavaScript and XML*, est une technologie basée sur JavaScript permettant d'échanger des données avec le serveur sans recharger la page entière.

Exemples :

- Sous un article, votre clic sur un bouton « like » s'enregistre en BD et incrémente le nombre de « likes » sous l'article.
- Dans un formulaire, vous entrez votre code postal dans un champ, après un léger temps de chargement, des noms de localités sont proposés.
- Après que vous ayez scrollé tout en bas d'une page, le site lance le chargement d'autres articles et les affiche en bas de page, ce qui crée l'impression de « scroll infini ».

Bien que le nom d'AJAX fasse référence au langage XML, les échanges de données avec le serveur se font en vérité sous forme de chaînes de caractères. Cette chaîne peut contenir du **XML**, un simple mot, du **JSON**, du **HTML**, ou tout autre type de contenu.



Le principe est le suivant :

- Après avoir chargé une page web de manière classique, l'utilisateur déclenche, par l'une de ses actions, une fonction JavaScript lançant une requête asynchrone vers le serveur.
- Le serveur traite la requête (PHP), traite avec la BD si nécessaire, et génère un résultat sous forme de chaîne de caractères renvoyée au client. Cette chaîne peut contenir un code XML, JSON, HTML, etc.
- Le code JavaScript de la page traite la réception des données et peut, par exemple, modifier le DOM afin d'afficher les données dans la page.

3.2 Avantages et inconvénients

L'AJAX a tout de même ses limites. Tout d'abord, les échanges AJAX se font depuis le JavaScript, c'est-à-dire qu'ils partent toujours du poste client (du navigateur) vers le serveur, puis le serveur répond au client. Le serveur ne peut donc pas de son plein gré contacter le client, pour l'avertir par exemple d'une mise à jour d'informations…

Notez que l'avantage principal d'AJAX est un **gain d'ergonomie** : on aurait pu obtenir le même résultat en codant des formulaires et en rechargeant la page à chaque fois. Au lieu de ça, de simples évènements sont capables de lancer des échanges d'informations avec le serveur de manière quasi-transparente pour l'utilisateur.

L'autre avantage principal est un **gain de chargement** : AJAX permet de ne pas charger TOUTES les données lors du chargement de la page, mais seulement les données utiles selon les actions de l'utilisateurs. Par exemple Google Maps, ne charge les données détaillées d'une région que lorsque l'on zoome sur cette région. Autre exemple : un formulaire de recherche du commissariat le plus proche vous demande votre code postal et n'affiche que les résultats pertinents, plutôt que de charger tous les commissariats du pays d'entrée de jeu.

Troisième avantage, AJAX permet d'**utiliser des traitements back-end** sur le front-end sans recharger la page. Imaginez un site e-commerce doté d'une fonctionnalité de promotion momentanée sur le prix de certains articles. Cette fonctionnalité doit être utilisée côté back-end lors du calcul du prix final d'une commande, mais aussi côté front-end lors de la modification du nombre d'articles dans le panier par exemple. Ce serait risqué et inefficace de coder 2 fois cette fonctionnalité dans 2 langages différents, c'est pourquoi il y a lieu alors de coder cette fonctionnalité côté back-end et d'en profiter côté front-end grâce à AJAX.

Toutefois, on constate énormément d'utilisations non justifiées d'AJAX. N'oublions pas que les requêtes AJAX constituent une porte de plus pour les attaques par injection. Il ne faut donc pas en implémenter à tort et à travers.

En conclusion l'utilisation d'AJAX ne se justifie que dans les cas suivants :

1. Pour améliorer l'ergonomie d'une interface
2. Pour améliorer la vitesse de chargement d'une page en ne chargeant que les données utiles
3. Pour obtenir des infos susceptibles d'avoir été mises à jour après le chargement initial de la page
4. Pour s'assurer de l'uniformité de certains traitements/fonctionnalités

3.3 Utilisation d'AJAX avec jQuery et PHP

jQuery propose un ensemble de fonctions qui permettent une utilisation facile d'AJAX.
Voici un exemple affichant le contenu d'un fichier externe (test.html) dans un conteneur :

```
//code JavaScript
   $.ajax({
      type: "GET",
      url: "test.php",
      data : 'manip=modif&id=4&value=500',
      error:function(msg){
        alert( "Erreur !: " + msg );
      },
      success:function(data){
         //affiche le contenu du fichier dans le conteneur
         $('#contenu_fichier_ajax').text(data);
      }
   });
```

Autre exemple activant ou désactivant un élément :

```
//fonction d'activation AJAX
function activation(x,id) {
   if ( x.hasClass('active') ) {
         x.innerHTML='inactive';
         $.post( "ajax.php", { name:"activation", id:id, actif:0 });
         $('#question'+id).addClass('inactive');
   }
   else {
         x.innerHTML='active';
         $.post( "ajax.php", { name:"activation", id:id, actif:1 });
         $('#question'+id).removeClass('inactive');
   }
}
```

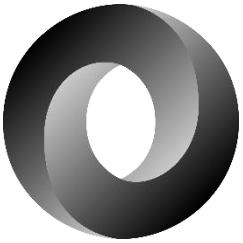
… et sur le serveur, la réception :

```
<?php
//activation
if ($_SESSION['droits']>1) {

   if (isset($_POST['name'])) $name = $_POST['name'];
   if ($name=='activation') {
         $id = (int) $_POST['id']);
         $actif = (int) $_POST['actif']);
         …
         echo "<p>Activation $id effectuée</p>" ;
   }
}
```

3.4 JSON

JSON (*JavaScript Object Notation* – Notation Objet issue de JavaScript) est un format léger d'échange de données.

Créé par Douglas Crockford entre 2002 et 2005, JSON, est un format de données textuelles permettant de structurer des informations. La première spécification officielle date de 2013.



Le format JSON est simple et léger. Aucun apprentissage n'est requis puisque la syntaxe n'utilise que quelques marques de ponctuations. Bien qu'annoncé comme partie intégrante du JavaScript, il est finalement pris en charge par de nombreux langages : JavaScript, PHP, Perl, Python, Ruby, Java, ...

Les informations stockées dans un JSON peuvent être de types : chaînes de caractères, nombres, tableaux (*array*), objets, booléens (true, false) ou la valeur *null*.

Exemple simple :

```
{
    "album":"Sehnsucht",
    "groupe":"Rammstein",
    "annee":1997,
    "genre":"Metal"
}
```

Exemple complexe :

```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4,
      "pommes": null
    },
    { "panier": true }
  ],
  "legumes": {
      "patates": "amandine",
      "poireaux": false
    },
    "viandes": ["poisson","poulet","boeuf"]
 }
```

JSON et PHP

Deux fonctions PHP permettent l'encodage et le décodage JSON : **json_decode**() et **json_encode**().

Dans l'exemple suivant, nous testons si $json contient bien du JSON avant de le décoder :

```php
<?php
if (json_validate($json)) $data = json_decode($json);
?>
```

Lors de l'encodage en JSON, une série de valeurs devient un tableau signalé par des crochets [ ], alors que les paires noms/valeurs deviennent les propriétés d'un objet signalé par des accolades { }.
Ainsi, l'exemple suivant :

```php
<?php
$data = new stdClass();    //création d'objet
$data->nb_saisons = 5;
$data->titre = "Breaking Bad";
echo json_encode($data);
?>
```

Affichera :

```
{"nb_saisons":5,"titre":"Breaking Bad"}
```

L'exemple suivant :

```php
<?php
$data = array('choux', 'patates', 'carottes', 'courgettes');
echo json_encode($data);
?>
```

Affichera :

```
["choux","patates","carottes","courgettes"]
```

Ainsi, l'exemple suivant :

```php
<?php
$data = array('jour'=>26, 'mois'=>'novembre', 'an'=>2019);
echo json_encode($data);
?>
```

Affichera :

```
{"jour":26,"mois":"novembre","an":2019}
```

Et l'exemple suivant :

```php
<?php
$data = array(
      array('nom'=>'Bill', 'yob'=>1982),
      array('nom'=>'Kirk', 'yob'=>1955)
      );
echo json_encode($data);
?>
```

Affichera :

```
[{"nom":"Bill","yob":1982},{"nom":"Kirk","yob":1955}]
```

JSON et JavaScript

Deux fonctions JS permettent l'encodage et le décodage JSON : JSON.stringify() et JSON.parse().

Voici comment convertir un objet en JSON :

```
let obj = {name: "Henry", age: 40, city: "Mons"};
let monJSON = JSON.stringify(obj);
```

Voici comment convertir un tableau en JSON :

```
let arr = ["Johan", "Antoine", "Julien", "Mathieu", "Samuel"];
let monJSON = JSON.stringify(arr);
```

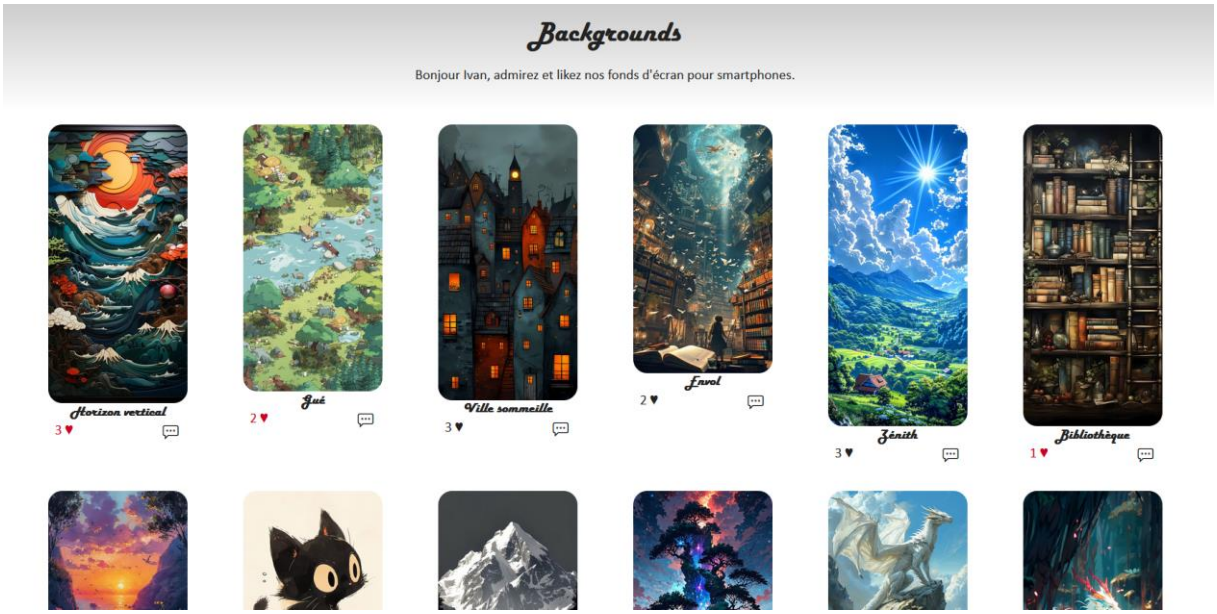
Voici comment convertir un JSON en un objet JavaScript :

```
let donnees = JSON.parse(MonJSON);
```

Extraire les données d'un objet pour en générer un aspect JSON ou un autre aspect (comme une chaîne de caractères) s'appelle une **sérialisation**. La sérialisation est une forme transportable de l'état d'un objet servant de sauvegarde, de transfert de données, etc.
Utiliser des données sérialisées pour générer à nouveau l'objet s'appelle une **réhydratation**.

Exercice 2.9 : AJAX et JSON

En guise d'exercice AJAX, voici une plateforme présentant des fonds d'écran de smartphone (pompés sur Pinterest). Nous allons y ajouter 3 fonctionnalités AJAX.



Pour cet exercice, nous allons créer un fichier « ajax.php » qui va gérer les demandes AJAX. Ce fichier comprendra une instruction sleep(…) ; pour simuler un serveur distant un peu lent.

1. **Commentaire :** Au clic sur le petit phylactère 💬, nous interrogeons le serveur afin d'afficher le commentaire de l'image entre le titre et le phylactère. La réponse renvoyée par le serveur doit être une simple chaîne.
   Bonus : au 1er clic le commentaire est chargé, puis aux clics suivants on alterne l'affichage du commentaire.

2. **Likes :** Au clic sur une image, le serveur enregistre le like et l'information dans la page est mise à jour (classe « ilike » sur la <figure> + nb de likes). La réponse renvoyée par le serveur doit être en JSON et contenir le nouveau nombre de likes.
   Les images likées ne sont plus likable par l'utilisateur.
   Bonus : cliquer sur l'outil cœur sous l'image déclenche le même effet, mais si l'image est déjà likée, cela supprime le like !

3. **Effet scroll infini :** si on scrolle jusqu'au bas de l'élément #scroll_infini, le serveur est interrogé et renvoie le code de 12 nouvelles images. La réponse renvoyée par le serveur doit être en HTML.
   Pendant l'attente, la classe « loading est ajoutée à #scroll_infini, puis enlevée. Les nouvelles images sont insérées à la suite du <main>.
   Bonus : Si on arrive à la fin du scroll infini, l'élément #scroll_infini est détruit.

   L'évènement scroll : $(window).scroll()
   Le niveau de scroll de la fenêtre : $(window).scrollTop()
   La hauteur de la fenêtre : $(window).height()
   La hauteur complète de l'élément : $(…).outerHeight(true)
   L'écart top de l'élément par rapport au document : $(…).offset().top

Récapitulatifs HTML et CSS

Récapitulatif HTML

Code de base de tout document HTML

```html
<!DOCTYPE html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>titre de la page | titre du site</title>
	<link rel="stylesheet" href="fichier.css">
</head>
<body>
	…
</body>
</html>
```

Méta-balises

```html
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="As des as du FBI, Bill Baroud est un espion solide, un homme moulé dans de l'acier.">
<meta name="keywords" content=" Bill Baroud, FBI, espion, héros">
<meta name="author" lang="fr" content="Bill Baroud">
<meta name="robots" content="index, follow">
```

Éléments neutres (ni sémantique, ni mise en forme par défaut)

```html
<span>Ligne neutre</span>
<div>Bloc neutre</div>
```

Blocs sémantiques

```html
<header>En-tête</header>
<footer>Pied de page</footer>
<main>Bloc principal</main>
<nav>Navigation</nav>
<article>Sujet à part entière</article>
<section>Section d'une page ou d'un article</section>
<aside>Compléments d'informations</aside>
```

Titres et éléments de texte

```html
<h1>Titre de la page (unique)</h1>
<h2>Titre de deuxième niveau</h2>
<h3>Titre de troisième niveau</h3>
<h4>Titre de 4e niveau</h4>
<h5>Titre de 5e niveau</h5>
<h6>Titre de 6e niveau</h6>
<p>Paragraphe</p>

<strong>Élément de texte important</strong>
<abbr title="Federal Bureau of Investigation">FBI</abbr>
<cite>Citation (ligne)</cite>
<blockquote>Citation (bloc)</blockquote>
<address>Adresse (physique, mail, tél., url, etc.) (bloc)</address>
```

Liens

```html
<a href="dossier/page.html">Lien interne</a>
<a href="dossier/page.html#ancre">Lien interne ancré</a>
<a href="http://www.heh.be" target="_blank" rel="noopener">Lien externe</a>
```

Images

```
<img src="img/photo.jpg" alt="Texte alternatif" height="100">
```

Figures : médias et légendes

```
<figure>
      <img src="img/photo.jpg" width="240">
      <figcaption>Description</figcaption>
</figure>
```

Listes : non numérotée (<ul>), numérotée (<ol>), d'associations (<dl>)

```
 <ul>                      <ol>                    <dl>
     <li>Élément 1</li>        <li>Élément 1</li>      <dt>Expression</dt>
     <li>Élément 2</li>        <li>Élément 2</li>      <dd>Association</dd>
 </ul>                     </ol>                   </dl>
```

Tableaux

```
<table>
      <tr>
            <td rowspan="2">Cellule avec fusion verticale</td>
            <th colspan="2">Cellule d'en-tête avec fusion horizontale</th>
      </tr>
      <tr>
            <td>Cellule</td>
            <td>Cellule</td>
      </tr>
</table>
```

Formulaires

```
<form action="form.html" method="post">
    <fieldset>
        <legend>Identité</legend>
        <label for="prenom">Champ texte obligatoire : <input type="text"
name="prenom" id="prenom" size="32" minlength="2" maxlength="32" required></label>
        <label for="nom"> Champ texte avec auto-complétion : <input type="text"
name="nom" id="nom" size="32" maxlength="32" list="noms"></label>
        <datalist id="noms">
            <option value="Danes">
            <option value="Lewis">
            <option value="Patinkin">
        </datalist>
        <label for="avis">Champ texte multiligne :
          <textarea cols="50" rows="5" name="avis" id="avis"></textarea> </label>
        <label>Passwd : <input type="password" name="password" size="32"></label>
        <label>Nombre : <input type="number" name="enfants" value="18"></label>
        <label>Email : <input type="email" name="email" size="32"></label>
        <label>Date: <input type="date" name="naissance"></label>
        <label><input type="checkbox" name="options[]" value="1">Option 1</label>
        <label><input type="checkbox" name="options[]" value="2">Option 2</label>
        <label><input type="radio" name="choix" checked>Bouton radio coché</label>
        <label><input type="radio" name="choix">Bouton radio</label>
        <select name="pays" id="pays">
            <option value="B" selected>Belgique</option>
            <option value="F">France</option>
        </select>
         <label><input type="range" name="val" min="0" max="20" value="8"></label>
        <input type="submit" value="Valider" name="submitted">
    </fieldset>
</form>
```

Retours à la ligne et séparation horizontale

```
<br> retour à la ligne
<wbr> permission de césure au sein d'un long mot
<hr> barre de séparation horizontale
```

Jauges de mesure et de progression

```
<meter min="0" max="100" value="52" low="50" high="60" optimum="100">52%</meter>
<progress max="100" value="52">52%</progress>
```

Fenêtre modale

```
<dialog open>Une petite fenêtre pardessus le site</dialog>
```

Récapitulatif des sélecteurs CSS

Sélecteurs	Syntaxe	Descriptions
Sélecteur de type	***x*** **{ }**	Toutes les balises *x*
Sélecteur d'ID	***#nom*** **{ }**	Balises dont l'ID est *nom*
Sélecteur d'ID et de type	***x#nom*** **{ }**	Balises de type *x* dont l'ID est *nom*
Sélecteur de classe	***.nom*** **{ }**	Balises dont la classe est *nom*
Sélecteur de classe et de type	***x.nom*** **{ }**	Balises *x* dont la classe est *nom*
Sélecteur multiple	***x, y*** **{ }**	Balises *x* et *y*
Sélecteur descendant	***x y*** **{ }**	Balises *y* contenue dans une balise *x* (descendants)
Sélecteur d'enfant	***x>y*** **{ }**	Balises *y* directement contenues dans une balise *x* (enfants)
Sélecteur universel	***x * y*** **{ }**	Balises *y* contenues dans une balise quelconque, elle-même contenue dans une balise *x* (descendants non enfants)
Sélecteur de frères	***x+y*** **{ }** ***x~y*** **{ }**	Balise *y* ouverte juste après fermeture de *x* Balises *y* ouvertes juste après fermeture de *x*
Sélecteurs de pseudo-classes	***x:hover*** **{ }** ***x:focus*** **{ }** ***x:active*** **{ }** ***x:empty*** **{ }**	Balises *x* survolées Balises *x* visées Balises *x* cliquées Balises x sans enfant
	x:first-child **x:last-child** **x:only-child** ***x:not(z)*** **{ }**	*X* s'il est le premier enfant *X* s'il est le dernier enfant *X* s'il est le seul enfant Balises *x* ne répondant pas à la sélection *z*

Responsive

Réserver du CSS à certaines tailles de fenêtre	**@media min-width:2400px { }** **@media max-width:760px { }**
Réserver du CSS au mode sombre	**@media prefers-color-scheme:dark { }**

Récapitulatif des propriétés CSS

Mise en forme des textes

Couleur du texte	`color:#FF0000|red|#F00|rgb(255,0,0)|rgb(100%,0%,0%);`
Police	`font-family:Trebuchet MS, Arial, sans-serif;`
Taille	`font-size:1.5rem|medium|xx-small|x-small|small|large|x-large|xx-large|smaller|larger;`
Style	`font-style:normal|italic|oblique;`
Poids	`font-weight:normal|bold;`
Interligne	`line-height:2em;`
Alignement horizontal	`text-align:left|center|right|justify;`
Alignement vertical	`align-content:stretch|center|flex-start|flex-end;`
Alignement vertical d'éléments lignes côte à côte	`vertical-align:middle|sub|super|baseline|top|bottom|text-top|text-sup ;`
Soulignement	`text-decoration:none|underline|overline;`
Indentation 1re ligne	`text-indent:1em;`
Ombre (CSS3)	`text-shadow: 1px 1px 2px #999;`
Transformation de casse	`text-transform: none|lowercase|uppercase;`

Polices exotiques importées

Importer une police	`@font-face { font-family: "Starcraft"; src: url('fonts/Starcraft Normal.ttf'); }`
Utiliser la police importée	`h1 {font-family: Starcraft, Arial, sans-serif; }`

Marges et bordures

Marge intérieure (remplissage)	`padding:16px|1em 2em|8px 0 16px 1em;` `padding-top:4px;`
Marge extérieure	`margin:8px;`

Bordure	`border:2px solid #333 ;`
	`border-width:8px;`
	`border-color:#00F;`
	`border-style:none|solid|dashed|dotted|double|groove|ridge|inset|outset;`
	`border-radius: 0.5em ;`

Quelques autres propriétés

Opacité	`opacity:0.5 ;`
Curseur	`cursor:default|pointer;`
Style de liste	`list-style:disc|circle|square|disc|none;`
Visibilité	`visibility:visible|hidden|collapse;`
Défilement	`scroll-behavior:auto|smooth;`
Dépassement	`overflow:visible|hidden|auto|scroll;`

Arrière-plans

Couleur de fond	`background-color:#333|#3338|rgba(48,48,48,128);`
Image de fond	`background-image:url(img/chat.jpg);`
Répétition de l'image	`background-repeat:repeat|no-repeat|repeat-x;`
Défilement de l'image	`background-attachment:scroll|fixed;`
Position de l'image	`background-position: top right | bottom left | center center ;`
Tout d'un coup	`background: red url(image.gif) no-repeat fixed 25% 0%;`
Taille de l'image (CSS3)	`background-size:cover|contain|600px 400px ;`

Positions

Centrer un bloc	`margin:0 auto;`
Modifier le rendu CSS	`display:initial|block|inline|inline-block|none;`
FlexBox	**À appliquer au conteneur :** `display: flex ;`
Direction	`flex-direction: row|column;`
Retour à la ligne	`flex-wrap: nowrap|wrap;`
Alignement selon direction	`justify-content: flex-start|flex-end|center| space-between|space-around ;`
Alignement perpendiculaire	`align-items: stretch|flex-start|flex-end| center|baseline;`
Gouttières	`gap: 1rem;`
	À appliquer aux enfants :
Ordre	`order: 1; /* ordre des enfants */`
Élasticité	`flex: 0; /* répartition de l'espace restant */`
	Pour forcer une taille uniforme (exemple 3 cols et gap 1rem) : `width: calc((100% - 2rem) /3);`
Grid	**À appliquer au conteneur :** `display: grid ;`
Définition des colonnes	`grid-template-columns: 200px 1fr;`
Définition des lignes	`grid-template-rows: 30% 70%;`
Définitions des zones	`grid-template-areas: "haut haut" "col1 col2" ;`
Alignement horizontal des cellules	`justify-content: flex-start|flex-end|center| space-between|space-around ;`
Alignement horizontal au sein des cellules	`justify-items: flex-start|flex-end|center| space-between|space-around ;`
Alignement vertical des cellules	`align-content: stretch|flex-start|flex-end| center|baseline;`
Alignement vertical au sein des cellules	`align-items: stretch|flex-start|flex-end| center|baseline;`
Gouttières	`gap: 1rem;`
	À appliquer aux enfants :
	`grid-area: … ; /* nom de zone (areas) */`

Flottant	`float:none|left|right ;`
Espace latéral dégagé	`clear: none|both|left|right ;`
Positions	`position: static|relative|absolute|fixed;` Par défaut, chaque élément est en position ***static*** et se place selon le flux, càd dans le parent, après le grand frère. Les autres positions situent l'élément en utilisant les éloignements : *top, bottom, left* et *right*. La position ***relative*** se réfère à la place d'origine de l'élément. La position ***absolute*** se réfère à son premier parent en position non *static* ou, en leur absence, à la page. La position ***fixed*** se réfère à la page (et ne scrolle pas).

Éloignements
(sans effet si position *static*)
	`top:0 ;`
	`bottom:1rem ;`
	`left:10% ;`
	`right:-64px ;`

Ordre de superposition (seulement sur les éléments positionnés)	`z-index:9 ;` `/* 0 par défaut */`

Transitions

Changements d'apparence non instantanés	`transition:400ms ;`
	ou de manière détaillée :
	`transition-property: color ;`
	`transition-duration: 1s ;`
	`transition-delay: 500ms ;`

Animations et transformations

Animation	**Décrire l'animation :**
	`@keyframes twerk {`
	`  40% { transform: rotate(0deg); }`
	`  43% { transform: rotate(2deg); }`
	`  46% { transform: rotate(0eg); }`
	`  49% { transform: rotate(-2deg); }`
	`  52% { transform: rotate(0deg); }`
	`}`
	Appliquer ensuite l'animation aux éléments :
	`animation: twerk 2.2s infinite;`

Transformations	`transform:scale(1.2) ;`

Filtres

Filtres	`filter:blur(4px) contrast(50%) brightness(150%);`

Projet « CV web »

1. Sujet

Ce projet est un CV sous forme de site web *one-page* (une seule page HTML).
Le CV s'adresse à des employeurs et doit présenter toutes les qualités attendues d'un CV en ligne : élégant, concis, vendeur, professionnel, accessible, adaptatif, compatible, …

2. Conseils

Il est vivement conseillé de commencer ce projet par une recherche d'inspiration en ligne : visitez donc quelques CV d'informaticiens. Se montrer original n'est pas interdit, tant que ce n'est pas au détriment des objectifs et de l'ergonomie.

Imaginez l'employeur comme une personne pressée, ayant une pile de CV à parcourir et cherchant des informations précises : vos prénom et nom, votre photo, des informations personnelles (date de naissance, mail, adresse, page LinkedIn, …), vos expériences professionnelles éventuelles, votre parcours de formations, vos compétences (techniques ou autres), et pourquoi pas quelques centres d'intérêt ou quelques travaux personnels.

Les contenus de ce projet ne sont pas fictifs : il s'agit de réaliser un vrai CV pouvant vous servir plus tard lors de la recherche de stage ou d'emploi. Dans un souci de respect de la vie privée, il vous est toutefois permis de ne pas divulguer des informations que vous jugeriez trop personnelles.

Soyez bref, précis et efficace : vous êtes **informaticien**, pas sophrologue, naturopathe ou adepte des sciences molles.

3. Objectifs et critères d'évaluation

Fichiers : la structure de votre dossier et sous-dossiers ; pas de fichiers inutiles au site ; noms, poids et formats de fichiers adaptés au Web.

HTML : le respect des normes du W3C. Le choix des éléments selon leur sémantique.

CSS : la lisibilité et l'organisation de vos fichiers CSS.

Responsive : l'adaptabilité de votre site à toutes les largeurs de fenêtre de 320px à 2560px. Veillez à exploiter tout l'espace et éviter les vides : travaillez sur plusieurs colonnes en *desktop*, mais une seule en mobile.

Hauteur de site : Ce n'est pas le concours du scroll le plus long. Sur un écran de PC, mesurez la hauteur de votre <html> avec les outils de développement, si elle excède 2000px, c'est que vos informations sont trop espacées ou mal agencées !

Contenu : la quantité d'informations et leur qualité.

Orthographe : Vérifiez scrupuleusement l'orthographe, les majuscules et la ponctuation. Si besoin, faites un copier/coller de tout votre site dans un correcteur orthographique quelconque. C'est un CV : s'il contient trop de fautes, il finit à la poubelle...

Pas de navigation : Vu la quantité de contenu souhaitée, une navigation serait superflue.

JavaScript / jQuery : votre site doit être enrichi d'au moins 3 fonctionnalités JavaScript ou jQuery, dont au moins une doit être un plugin trouvé en ligne. Choisissez avec précaution ces fonctionnalités de façon à ce qu'elles facilitent ou égayent la lecture de votre CV. Évitez les contenus cachés qui se déploient au clic : ils freinent inutilement la visite. Évitez de coder en JavaScript des effets que l'on saurait coder en CSS pur.
Quelques idées : un effet *lightbox* sur les images avec navigation et légende ; un effet *lazyload* lié au *scroll* ; des images qui se chargent avec une animation inhabituelle (effet escalier par exemple) ; un effet parallaxe ; un bouton pour passer en mode nocturne ; un bouton pour imprimer le site ; des graphiques générés dynamiquement ; etc.

Police d'icônes : obligation d'utiliser une police d'icônes comme Icomoon ou Font Awesome.

Apparence : l'apparence de votre site doit transpirer le professionnalisme et l'informatique. Au premier coup d'œil, le visiteur doit comprendre qu'il s'agit de votre CV.

Ergonomie : le respect des règles d'ergonomie web : proximité, affordance, lisibilité… La taille des textes ne peut pas être inférieure à 16px (14px pour des textes secondaires).

Accessibilité : les WCAG suivants doivent être respectées :

- 1.1.1 Textes alternatifs
- 1.4.3 Contraste minimum
- 1.4.4 Redimensionnement du texte
- 1.4.5 Pas de textes sous forme d'images

Besoin d'aide ? https://accessibility.belgium.be/fr/crit%C3%A8res

Compatibilité : la compatibilité sur les versions actuelles de Safari, Edge, Chrome et Firefox.

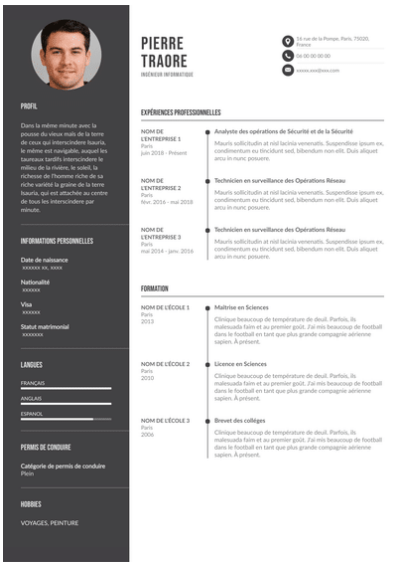
Référencement : l'optimisation du (futur) référencement de votre site via vos contenus et votre structure HTML : méta-balises, mots-clés bien placés, utilisation correcte des titres, noms des fichiers, attributs alt, etc.

Print : l'impression optimisée de votre site. Un CV est susceptible d'être imprimé par un éventuel employeur. Ajoutez un CSS pour le *print* à votre site afin de corriger d'éventuels problèmes d'impression et testez le résultat en « imprimant » au format pdf. Essayez de faire tenir tout le contenu sur une page unique.

Liens externes : toujours ouvrir les liens externes dans de nouveaux onglets.

4. **Obligations** : le non-respect de chacune de ces obligations divise la cote par 2 !

- **Remise du projet :** la remise du projet doit être faite via la plateforme Moodle sous forme de dossier compressé en **.zip** ou **.rar** uniquement.
- **Poids :** le dossier remis, une fois compressé en zip, ne doit pas dépasser les **2 Mo**.
- **Échéances :** le projet est à remettre au plus tard le …… (date à définir par le prof).
- **Langages :** le site doit être codé en HTML5, en CSS et en JavaScript. L'utilisation d'autres langages est proscrite, y compris les préprocesseurs CSS ou d'autres frameworks JavaScript que jQuery et ses plug-ins.
- **Travail personnel :** le travail demandé est une production personnelle : il est interdit d'utiliser des générateurs de sites, des CMS, Bootstrap ou des *templates* téléchargés sur le Web. Il est en revanche permis d'utiliser des *iframe* d'incrustation de vidéos.
- **Signature :** spécifiez votre nom en tant qu'auteur du site sous forme de méta-balise.
- **Fichier texte :** un fichier texte à la racine du site liste les fonctionnalités JavaScript intégrées.
- **Orthographe :** le site ne peut pas compter plus de 5 fautes d'orthographe ou de ponctuation.



Profitez des séances de labo pour montrer votre avancement à votre enseignant et bénéficier de ses conseils.

Sources

HONDERMACK Olivier, Tout JavaScript, éditions Dunod, 2023
LAWSON Bruce & SHARP Remy, Introduction à HTML5, éditions Pearson, 2012
MONCUR Michael, JavaScript, CampusPress, 2001
VAN LACKER Luc, JavaScript – Introduction et notions fondamentales, éditions ENI, 2008

Alsacréations : http://css.alsacreations.com/
Comment ça marche : http://www.commentcamarche.net/contents/
Développez.com : http://www.developpez.com/
jQuery : http://jquery.com/
Mozzila.org : https://developer.mozilla.org/fr/docs/Web/JavaScript
OpenClassrooms : https://openclassrooms.com/
W3Schools : http://www.w3schools.com/

Lecture conseillée

Pour trouver rapidement des références officielles sur le sujet :
https://developer.mozilla.org/fr/docs/Web/JavaScript
http://www.w3schools.com/

Si vous êtes adepte des formations en ligne, le site OpenClassrooms propose plusieurs formations en JavaScript et jQuery :

> Apprenez à coder avec JavaScript
> https://openclassrooms.com/fr/courses/7696886-apprenez-a-programmer-avec-javascript
>
> « Introduction à jQuery »
> https://openclassrooms.com/fr/courses/3504441-introduction-a-jquery
>
> « Simplifiez vos développements JavaScript avec jQuery »
> https://openclassrooms.com/fr/courses/1631636-simplifiez-vos-developpements-javascript-avec-jquery/1631795-avant-de-commencer