

Javascript

Romain GONÇALVES

Coding
Factory



Plan de cours

- I. Initialisation du Sprint
- II. Variables et opérateurs
- III. Structures Conditionnelles et Boucles
- IV. Ensembles de Données et Fonctions
- V. Fonctions utiles du langage
- VI. Le DOM
- VII. Programmation événementielle avec jQuery
- VIII. Fin du Sprint
- IX. Lab' Day
- X. Corrigés

Objectifs du cours

- Maîtriser la syntaxe du langage et ses concepts de base
- Découvrir le DOM et ses éléments
- Débuter la programmation événementielle avec jQuery



I. Initialisation du Sprint



Historique du Javascript

Année	Version
1997	ES1
1998	ES2
1999	ES3
Annulé	ES4
2009	ES5
2015	ES6 / ES 2015
2016	ES7 / ES 2016
2017	ES8 / ES 2017

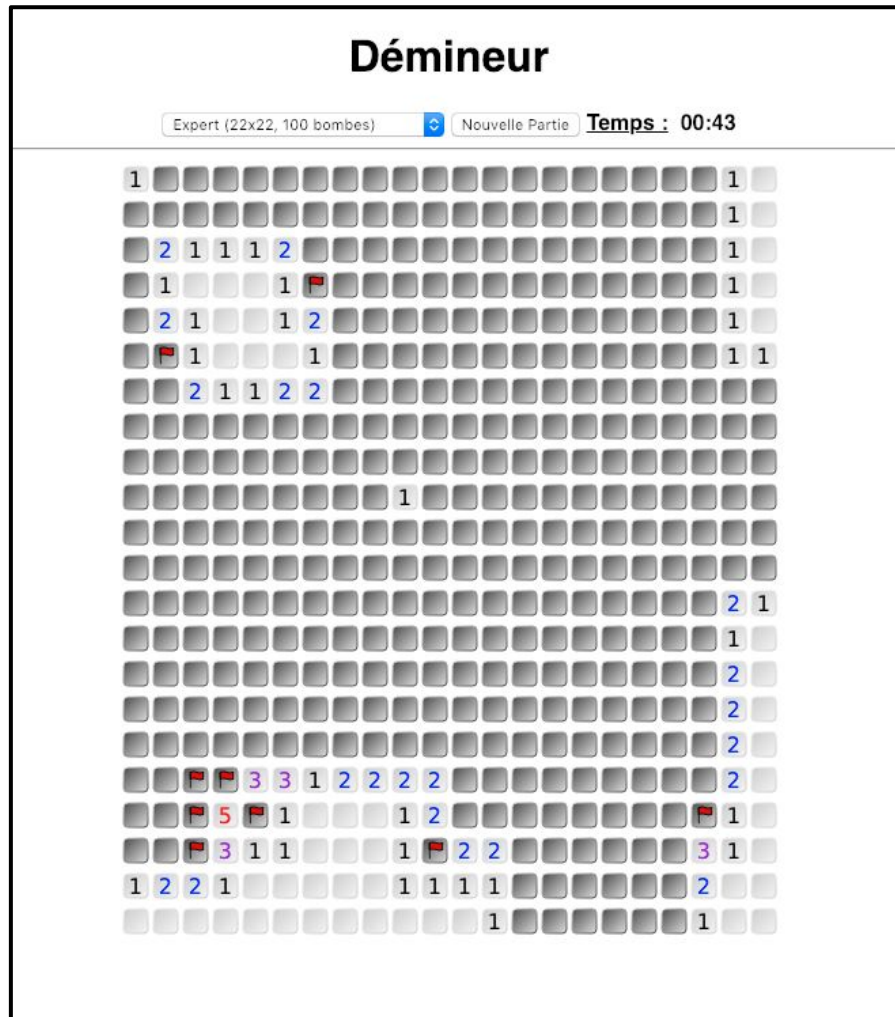
Compatibilité ES 5				
Chrome 23	Firefox 21	IE 9	Edge 10	Safari 6

Compatibilité ES 6			
Chrome 58	Firefox 54	Edge 14	Safari 10

Compatibilité ES 7	
Chrome 68	Opéra 47

Projet du Sprint

Réalisation d'un jeu de
démineur en HTML, CSS
et Javascript :



Backlog

- HTML
 - Sélection de la difficulté
 - Bouton de lancement de partie
 - Compteur de temps
- CSS
- Javascript
 - Lancement de la partie
 - Placement des bombes
 - Sélection d'une case
 - Révélation des cases adjacentes si vides
 - Clic droit pour placer/enlever un drapeau
 - Fin de jeu

Niveaux de difficulté :

- *Débutant : 9x9 cases, 10 bombes*
- *Intermédiaire : 16x16 cases, 40 bombes*
- *Expert : 22x22 cases, 100 bombes*
- *Maître : 30x30 cases, 250 bombes*

Environnement de Travail

- Visual Studio Code (ou équivalent : Sublime Text, Atom, Vim, ...)
- Navigateur (Google Chrome ou Firefox de préférence)
- Outils de développement du navigateur

Méthodologie de travail

- Créer un dossier pour les exercices (et plus tard un autre pour le projet)
- Ouvrir le dossier dans l'éditeur de code
- Dans le Finder, naviguer jusqu'au dossier
- Ouvrir le fichier HTML avec le navigateur (**clic-droit > Ouvrir avec** ou Drag&Drop)
- Pour ouvrir les outils de développement, faire `command+Option+J` sur l'onglet concerné

Exécution du code

Tout comme le CSS, le javascript peut être écrit dans le HTML, ou chargé depuis un fichier.

On notera qu'il est préférable que le Javascript soit chargé après le code HTML de la page afin d'améliorer l'expérience utilisateur.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Cours JS</title>

  <!-- Inclusion de CSS depuis un fichier -->
  <link rel="stylesheet" href="style.css" />

  <style>
    /* CSS directement dans le HTML */
  </style>
</head>

<body>
  <!-- Balises HTML-->
  <div></div>
  <!-- ... -->

  <!-- Inclusion de JS depuis un fichier -->
  <script type="text/javascript" src="fonctions.js"></script>

  <script type="text/javascript">
    // Code Javascript directement dans le HTML
    alert('Bonjour !');
    // ...
  </script>
</body>

</html>
```

Infos pratiques

- Si vous souhaitez tester un bout de code sans modifier votre script principal, vous pouvez utiliser l'onglet "Console" des outils de développements.
- Clavier Mac :
 - Accolades : `Alt + (` et `Alt +)`
 - Crochets : `Alt + Shift + (` et `Alt + Shift +)`
 - Barre verticale (pipe) : `Alt + Shift + |`
- La détection d'erreur de VS Code se recharge à la sauvegarde du fichier

II. Variables et Opérateurs

Les types de variables

- Numériques : **Number** (*le symbole décimal est le point, pas la virgule*)
- Textuels : **String**
- Logiques : **Boolean**
- Autres : **Date, Array, Object**

- Types spéciaux :
 - **null**
 - **undefined** (*variable déclarée, mais aucune valeur n'a jamais été affectée*)

- Le typage est **faible** (la variable change automatiquement de type en fonction de sa valeur et des opérations qui lui sont appliquées), et priorise les **chaînes de caractères** ($5 + \text{"5"} \Rightarrow \text{"55"}$)

Opérateurs

- Arithmétique :
 - Addition : +
 - Soustraction : -
 - Multiplication : *
 - Division : /
 - Modulo : %
 - Incrémentation : ++
 - Décrémentement : --
- Concaténation de String : +

Affichage d'une variable

- Pour afficher le contenu d'une variable en Javascript, trois grandes solutions existent :
 - Modifier le contenu du HTML (*conversion en String*)
 - Utiliser la fonction **alert(variable)**, qui l'affiche dans une pop-up (*conversion en String + bloque le navigateur tant qu'elle n'est pas fermée*)
 - Utiliser la fonction **console.log(variable1, variable2, ...)**, qui l'affiche dans l'onglet "Console" des outils de développement (*conserve le type*)

Syntaxe

- La déclaration est faite avec le mot-clé `var`, suivi du nom de la variable et éventuellement de sa valeur initiale



```
var variable1;  
var variable2 = "Hello",  
    variable3,  
    variable4;  
variable4 = "World !";  
console.log(variable1, variable2, variable3, variable4);  
// Affichera : undefined "Hello" undefined "World !"
```

- La déclaration sans le mot-clé `var` créera une variable globale ([slide 29](#))

Syntaxe

- Depuis ES6/ES2015, la déclaration est faite avec l'un des trois mots-clés :
 - `var` : assignation multiple, non scopée
 - `let` : assignation multiple, scopée
 - `const` : assignation unique, scopée

```
var variable1 = "foo";
let variable2 = "bar";
const variable3 = "baz";

if (true) {
  var variable1 = "toto";
  let variable2 = "tata";
  const variable3 = "titi";

  console.log(variable1, variable2, variable3);
  // Affichera : "toto" "tata" "titi"
}
console.log(variable1, variable2, variable3);
// Affichera : "toto" "bar" "baz"
variable3 = "test";
// Génèrera une erreur "Assignment to constant variable"
```



III. Structures Conditionnelles et Boucles



Opérateurs booléens

- En supposant deux booléens A et B, il y a quatre opérations possibles :
 - NOT (!) : **!A** sera vrai si et seulement si A est faux (et inversement)
 - AND (&&) : **A&&B** sera vrai si et seulement si les deux opérandes sont vraies
 - OR (||) : **A||B** sera vrai si et seulement si au moins une des deux opérandes est vraie
 - XOR (^) : **A^B** sera vrai si et seulement si exactement une des deux opérandes est vraie
- On peut en tirer les tables de vérité suivantes (V = Vrai, F = Faux) :

A	B	A&&B
F	F	F
V	F	F
F	V	F
V	V	V

A	B	A B
F	F	F
V	F	V
F	V	V
V	V	V

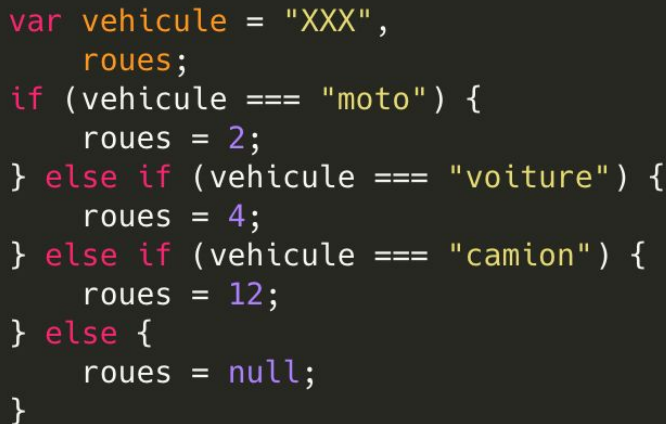
A	B	A^B
F	F	F
V	F	V
F	V	V
V	V	F

Comparaisons

- En supposant deux variables A et B, il existe des opérateurs de test retournant un booléen :
 - Différence ($!=$) : $A != B$ sera vrai si et seulement si A est différent de B (avec transtypage)
 - Différence stricte ($!==$) : $A !== B$ sera vrai si et seulement si A est différent de B (sans transtypage)
 - Égalité ($==$) : $A == B$ sera vrai si et seulement si A est égal à B (avec transtypage)
 - Égalité stricte ($===$) : $A === B$ sera vrai si et seulement si A est égal à B (sans transtypage)
 - Inférieur à ($<$)
 - Inférieur ou égal à ($<=$)
 - Supérieur à ($>$)
 - Supérieur ou égal à ($>=$)

Tests

Afin de pouvoir exécuter du code en fonction d'une condition, on utilise l'instruction `"if"`, accompagnée éventuellement des instructions `"else"` et/ou `"else if"`:



```
var vehicule = "XXX",  
    roues;  
if (vehicule === "moto") {  
    roues = 2;  
} else if (vehicule === "voiture") {  
    roues = 4;  
} else if (vehicule === "camion") {  
    roues = 12;  
} else {  
    roues = null;  
}
```

Tests

Dans le cas où il faut tester une même variable pour plusieurs valeurs, on préférera l'instruction "**switch**" :

```
var vehicule = "voiture"
switch (vehicule) {
  case "moto":
    roues = 2;
    break;
  case "voiture":
    roues = 4;
    break;
  case "camion":
    roues = 12;
    break;
  default:
    roues = null;
    break
}
```

Tests

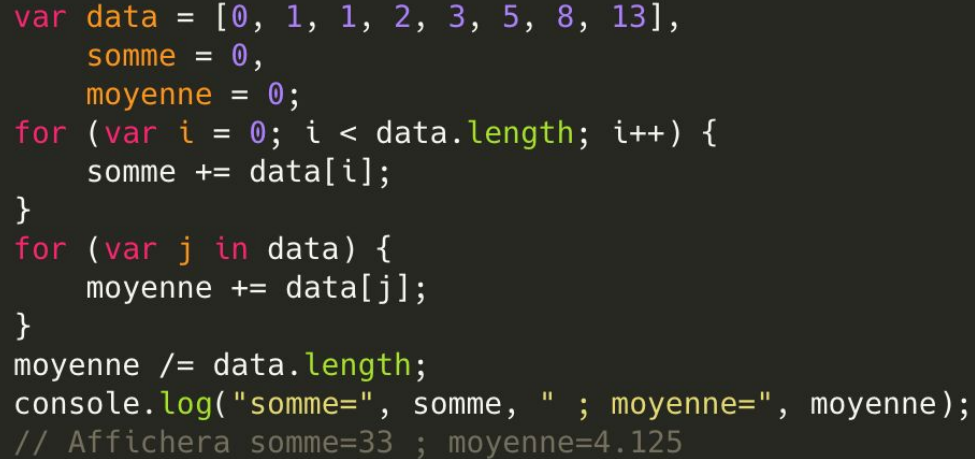
Dans le cas d'une condition simple permettant d'affecter l'une de deux valeurs à une même variable, on peut utiliser la condition ternaire :



```
var majeur = (age >= 18) ? "Vous êtes majeur" : "Vous êtes mineur";  
console.log(majeur + (age == 17 ? ", mais plus pour longtemps" : ""));
```

Syntaxe : (<CONDITION>) ? <VALEUR SI VRAI> : <VALEUR SI FAUX>

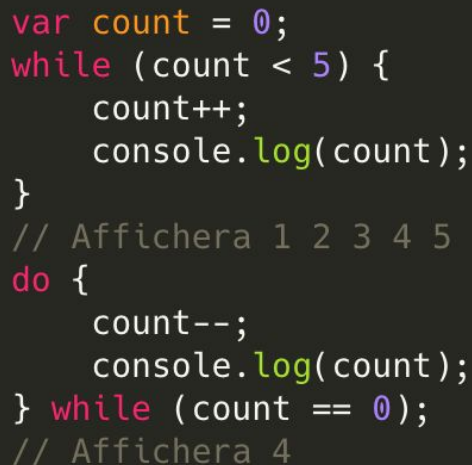
Les boucles for et for..in



```
var data = [0, 1, 1, 2, 3, 5, 8, 13],
    somme = 0,
    moyenne = 0;
for (var i = 0; i < data.length; i++) {
    somme += data[i];
}
for (var j in data) {
    moyenne += data[j];
}
moyenne /= data.length;
console.log("somme=", somme, " ; moyenne=", moyenne);
// Affichera somme=33 ; moyenne=4.125
```

Syntaxe : **for** (<DEPART?>;<SORTIE?>;<ITERATION?>) { <CODE> }
for (<VAR_CLE> in <ENSEMBLE>) { <CODE> }

Les boucles while et do..while



```
var count = 0;
while (count < 5) {
  count++;
  console.log(count);
}
// Affichera 1 2 3 4 5
do {
  count--;
  console.log(count);
} while (count == 0);
// Affichera 4
```

Syntaxe : **while** (<SORTIE?>) { <CODE> }

do { <CODE> } **while** (<SORTIE?>) ;




IV. Ensembles de Données et Fonctions




Les tableaux

- Les tableaux sont indexés numériquement
- La classe **Array** possède une propriété **length** qui contient le nombre d'éléments, accessible avec la syntaxe **tableau.length**



```
var data = ['a', 'b', 'c', 'd'];
console.log(data.length + " valeurs");
for (var index in data) {
    console.log(index + ":" + data[index])
}
// Affichera :
// 4 valeurs
// 0:a
// 1:b
// 2:c
// 3:d
```

Les fonctions



```
function bissextile(annee) {  
    return (annee % 4 == 0) && (annee % 100 != 0 || annee % 400 == 0);  
}  
console.log(bissextile(1988)); // True  
console.log(bissextile(2000)); // True  
console.log(bissextile(2018)); // False  
  
function moyenne() {  
    var somme = 0;  
    for (var i in arguments) {  
        somme = somme + arguments[i];  
    }  
    return somme / arguments.length;  
}  
console.log(moyenne(10, 20)); // 15  
console.log(moyenne(15, 10, 5, 10, 12, 14, 18)); // 12
```

Syntaxe : **function** <NOM> (<PARAMETRES?>) { <CODE> }

Le scope

Javascript est très permissif avec la portée des variables. Les fonctions ont accès en lecture (et parfois en écriture) aux variables existant en dehors de leur **“scope”** d'exécution.

```
français = 'Bonjour';
anglais = 'Hello';

var espagnol = 'Buenos dias';
var portugais = 'Bom dia';

function sayGoodbye() {
  var français = 'Au revoir';
  anglais = 'Goodbye';

  var espagnol = 'Hasta luego';
  portugais = 'Adeus';

  var italien = "Arrivederci";
  allemand = "Auf wiedersehen";
  console.log(français, anglais, espagnol, portugais, italien, allemand);
}

// italien et allemand ne sont pas encore définies
console.log(français, anglais, espagnol, portugais);
// Affichera : "Bonjour" "Hello" "Buenos dias" "Bom dia"

sayGoodbye();
// Affichera : "Au revoir" "Goodbye" "Hasta luego" "Adeus" "Arrivederci" "Auf wiedersehen"

// italien n'est plus définie
console.log(français, anglais, espagnol, portugais, allemand);
// Affichera : "Bonjour" "Goodbye" "Buenos Dias" "Adeus" "Auf wiedersehen"
```

Les objets

- Les objets sont des ensembles de données indexées par clé explicite :
- *On notera l'ordre alphabétique des clés dans la boucle*
- *Les clés sont converties en String, pas les valeurs*

```
var data = {  
    prop1: 'a',  
    prop2: 'b',  
    3: 'c',  
    'prop espace': 4  
};  
for (var index in data) {  
    console.log(index + ":" + data[index])  
}  
// Affichera :  
// 3:c  
// prop1:a  
// prop2:b  
// prop espace:4
```

Les objets

- La classe **Object** est la classe racine de la programmation orientée objet qui permet de définir des objets, avec leurs propriétés et méthodes :

```
/* Version ES5 */
function User(name) {
    this.name = name;
}
User.prototype.sayHi = function () {
    return "Bonjour, je m'appelle " + this.name;
}
const me = new User("Romain");
console.log(me.sayHi());
// Affichera "Bonjour, je m'appelle Romain"
```

```
/* Version ES6 */
class User {
    constructor(name) {
        this.name = name;
    }

    sayHi() {
        return "Bonjour, je m'appelle " + this.name;
    }
}
const me = new User("Romain");
console.log(me.sayHi());
// Affichera "Bonjour, je m'appelle Romain"
```



V. Fonctions utiles du langage



Fonctions de la classe String

<code>"Hello World".length //=> 11</code>	Contient la longueur de la chaîne (<u>propriété</u>)
<code>"Hello World".charAt(6) //=> "W"</code>	Retourne le caractère à l'index désigné
<code>"Hello World".split(" ") //=> ["Hello", "World"]</code>	Découpe une chaîne selon un séparateur
<code>"*".repeat(5); //=> "*****"</code>	Répète la chaîne le nombre de fois donné
<code>"piou piou".toUpperCase() //=> "PIOU PIOU"</code> <code>"BOOM".toLowerCase() //=> "boom"</code>	Changement de casse
<code>"Hello World".substring(4, 9) //=> "o Wo"</code> <code>"Hello World".substring(6) //=> "World"</code>	Retourne la sous-chaîne comprise entre les deux index (ou à partir du premier)
<code>" Hello world! ".trim() //=> "Hello world!"</code>	Supprime les espaces au début et à la fin

Propriété et Fonctions de la classe Array

<code>[1, 2, 3].length // => 3</code>	Compte le nombre d'éléments (<u>propriété</u>)
<code>[1, 2, 3].indexOf(3) // => 2</code>	Retourne l'index d'un élément (ou -1)
<code>["a", "b", "c"].includes("b") //=> true</code>	Teste la présence d'une valeur
<code>["a", "b"].values() // => ["a", "b"]</code> <code>["a", "b"].keys() // => [0, 1]</code>	Extrait les clés/valeurs
<code>[1, 30, 4, 20].sort() // => [1, 20, 30, 4]</code>	Trie le tableau (<u>comparaison en String</u>)
<code>[1, 2, 3].join(",") // => "1,2,3"</code>	Calcule l'intersection de 2 tableaux
<code>[1, 2].concat([3, 4]) // => [1, 2, 3, 4]</code>	Fusionne 2 tableaux (<u>sans modifier</u>)
<code>[1, 2].push(3) // => [1, 2, 3]</code> <code>[1, 2, 3].pop() // => [1, 2] et retourne 3</code>	Ajoute/retire un élément en fin de tableau (<u>modifie le tableau initial</u>)
<code>[2, 3].unshift(1) // => [1, 2, 3]</code> <code>[1, 2, 3].shift() // => [2, 3] et retourne 1</code>	Ajoute/retire un élément en début de tableau (<u>modifie le tableau initial</u>)

Fonctions de la classe Array

```
function strlen(str){ return str.length; }  
["Hello", "World"].map(strlen); //=> [5, 5]
```

```
[1, 2, 3].map(function(e1) {  
    return e1*2;  
}); //=> [2, 4, 6]
```

Retourne un tableau contenant les éléments du tableau en paramètre, auxquels on a appliqué la fonction donnée

```
[1, 2, 3, 4, 5].reduce(function(total, val) {  
    return total+val;  
}, 0) //=> 15
```

Réduit un tableau à une valeur unique en appliquant successivement la fonction aux paramètres et en reprenant le résultat à chaque appel (**total**). Le troisième paramètre correspond à la valeur initiale de **total** (NULL par défaut)

Fonctions de la classe Math

<code>Math.abs(-1) // => 1</code>	Valeur absolue d'un nombre
<code>Math.round(Math.PI) //=> 3</code> <code>Math.floor(Math.PI) //=> 3</code> <code>Math.ceil(Math.PI) //=> 4</code>	Arrondit (à l'entier le plus proche, à l'entier inférieur et à l'entier supérieur)
<code>Math.min(1, 10, 100) //=> 1</code> <code>Math.max(1, 10, 100) //=> 100</code>	Retourne la plus petite (grande) valeur parmi les paramètres passés
<code>Math.pow(2, 10) //=> 1024</code>	Calcule le nombre à la puissance du deuxième
<code>Math.random() //=> 0.2860409871038556</code>	Retourne un float pseudo-aléatoire entre 0 (inclus) et 1 (exclus)
<code>Math.cos(angle) // Math.sinh(angle)</code> <code>Math.atan(angle) // ...</code>	Fonctions trigonométriques, hyperboliques et arcs

Fonctions de la classe Date

<code>Date.now()</code> //=> 1547752752642	Retourne le timestamp en millisecondes
<code>new Date();</code> //=> Thu Jan 17 2019 20:20:06 GMT+0100 <code>new Date("2000-01-01");</code> //=> Sat Jan 01 2000 01:00:00 GMT+0100 <code>new Date("2000");</code> //=> Sat Jan 01 2000 01:00:00 GMT+0100 <code>new Date(2000, 0, 1)</code> //=> Sat Jan 01 2000 01:00:00 GMT+0100	Crée une instance avec la date/heure actuelle ou celle spécifiée (<u>Mois indexés à zéro</u>)
<code>var d = new Date();</code> <code>d.getFullYear()</code> <code>d.setFullYear(2000);</code> <code>d.getMonth()</code> <code>d.setMonth(0);</code> <code>d.getDay()</code> <code>d.setDay(1);</code> <code>...</code> <code>...</code>	Retourne ou modifie un des paramètres de date/heure (11 paramètres, disponibles en fuseau relatif ou absolu)
<code>d.toUTCString();</code> //=> Sat Jan 01 2000 01:00:00 GMT+0100	Convertit une Date en chaîne

Exercice 1 :

Calculer et afficher les 50 premières valeurs de la suite de Fibonacci à l'aide d'un tableau

$$u_0 = 0, u_1 = 1, u_n = u_{n-1} + u_{n-2} \Rightarrow 0, 1, 1, 2, 3, 5, 8, \dots$$

Exercice 2 :

Créer une fonction qui simule un lancer de dé (retourne un nombre aléatoire entre 1 et 6).

Tester en affichant 10 lancers

Exercice 3 :

Combien de lancers de dé faut-il faire pour que toutes les faces soient sorties une fois ?

- Créer une fonction qui teste et retourne le nombre
- Créer une fonction qui simule X tests et retourne la moyenne (vérifier jusqu'à $X=10\ 000$)

Exercice 4 : Le lièvre et la tortue

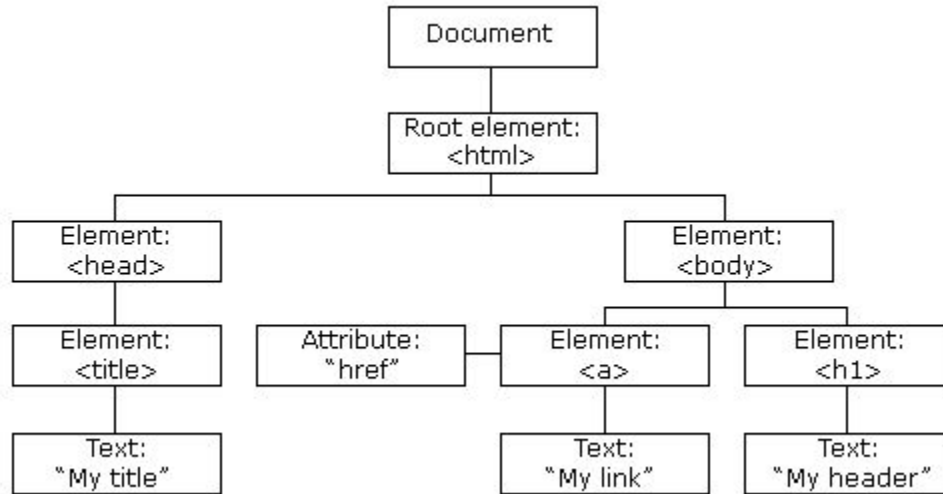
On lance un dé : si on trouve 6, le lièvre gagne, sinon, la tortue avance d'une case. Le parcours fait 6 cases.

- Créer une fonction qui retourne le gagnant d'une course
- Créer une fonction qui simule X courses et retourne les pourcentages de victoire (vérifier jusqu'à $X=10\ 000$)

VI. Le DOM

Le DOM (Document Object Model)

- Interface de programmation pour les documents HTML
- Permet d'interagir avec la structure de la page (HTML) et son apparence (CSS)



Le DOM (Document Object Model)

Le DOM est composé de deux objets principaux liés :

- **window** est l'objet racine, qui représente la fenêtre (ou l'onglet) du navigateur et permet :
 - L'affichage de messages (`window.alert("Message")`), de demandes de saisie (`window.prompt("Saisir un nombre :")`) et de confirmations (`window.confirm("Etes-vous sur ?")`)
 - La fermeture de l'onglet (`window.close()`) ou le déclenchement de l'impression (`window.print()`)
 - Le changement d'URL (`window.location`) et la gestion de l'historique (`window.history`)
 - La récupération des dimensions de la page (`window.innerWidth` et `window.innerHeight`) et de l'état du défilement de page (`window.scrollX` et `window.scrollY`)
 - La gestion de timers (actions retardées ou répétées)
 - La gestion des événements ([slide 53](#))
- **document** est l'objet enfant, qui représente la structure HTML/CSS de la page et permet :
 - La navigation dans l'arbre de balises HTML (objets `HTMLElement`)
 - L'ajout, modification et suppression des balises et leurs attributs (dont le style et le texte)

Messages

Fonctions :

`alert(<message>)`

`prompt(<message>, <valeur_par_defaut?>)`

`confirm(<message>)`



A screenshot of a JavaScript prompt dialog box. The title bar is not visible. The main text area contains the message "Cette page indique" followed by "Saisir un nombre". Below this is a text input field containing the number "5". At the bottom right, there are two buttons: "Annuler" (disabled) and "OK" (active).

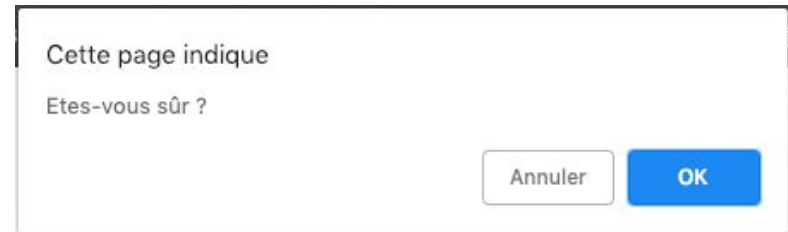
Affiche un message (ne retourne rien)

Demande une saisie (retourne " " ou la saisie)

Demande une confirmation (retourne true / false)



A screenshot of a JavaScript alert dialog box. The title bar is not visible. The main text area contains the message "Cette page indique" followed by "Bonjour". At the bottom right, there is a single "OK" button.



A screenshot of a JavaScript confirm dialog box. The title bar is not visible. The main text area contains the message "Cette page indique" followed by "Etes-vous sûr ?". At the bottom right, there are two buttons: "Annuler" (disabled) and "OK" (active).

Gestion de l'URL

L'objet `window.location` permet de lire et modifier l'URL via les propriétés :

<code>href</code>	URL entière
<code>hash</code>	Hash (partie qui suit le symbole “#” inclus)
<code>protocol/host/hostname/port</code>	Protocole, Hôte et Port, Hôte, Port
<code>search</code>	Paramètres GET (partie qui suit le symbole “?” inclus)

Il possède également les méthodes suivantes :

<code>assign(url)</code>	Naviguer vers une nouvelle page
<code>reload(ignoreCache)</code>	Recharger la page (en ignorant le cache ou non)
<code>replace</code>	Naviguer vers une nouvelle page (sans enregistrer la page actuelle dans l'historique)

`window.location.assign("https://www.google.com");` est équivalent à
`window.location = "https://www.google.com";`

Timers : actions retardées

Fonctions :

`setTimeout(<fonction>, <delai_ms>)` Retourne une référence au timer et le démarre
`clearTimeout(<reference>)` Arrête un timer désigné



```
var timer = setTimeout(function () {  
    console.log("plus tard");  
}, 3000);  
console.log("maintenant");  
// Affichera "maintenant" immédiatement  
// et "plus tard" 3 secondes après  
// Peut être interrompu en exécutant :  
// clearTimeout(timer);
```

Timers : actions répétées

Fonctions :

`setInterval(<fonction>, <delai_ms>)`

Retourne une référence au timer et le démarre

`clearInterval(<reference>)`

Arrête un timer désigné



```
var timer = setInterval(function () {  
    console.log("plus tard");  
}, 3000);  
console.log("maintenant");  
setTimeout(function () {  
    clearInterval(timer);  
    console.log("fin");  
}, 15000);  
// Affichera "maintenant" immédiatement et "plus tard" 3 secondes après et toutes les 3 secondes  
// Au bout de 15 secondes, affiche "fin" (5 affichages de "maintenant")  
// Peut être interrompu en exécutant :  
// clearInterval(timer);
```


Parcours et manipulations du document

En supposant la structure HTML ci-dessous, voici ce qu'il est possible de faire :

```
<!DOCTYPE html>
<html>

<head>
  <title>Accueil</title>
</head>

<body>
  <header>
    <h1>Bienvenue</h1>
    <nav>
      <ul>
        <li><a href="accueil">Accueil</a></li>
        <li><a href="contact">Contact</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <section class="text">
      <p id="paragraph1">
        <u>Titre :</u> Contenu
      </p>
    </section>
    <section class="text">
      <p id="paragraph2">
        <u>Titre :</u> Contenu
      </p>
    </section>
  </main>
</body>

</html>
```

```
console.log(document.children); // HTMLCollection(1) [<html>]
console.log(document.children[0].firstChild); // <head>
console.log(document.children[0].children[1].children); // HTMLCollection(2) [<header>, <main>]
console.log(document.getElementsByTagName('a')); // HTMLCollection(2) [<a>, <a>]
console.log(document.getElementsByClassName('text')); // HTMLCollection(2) [<section>, <section>]
console.log(document.getElementById('paragraph1')); // <p>
console.log(document.querySelector('.text > p#paragraph1 u')); // <u>
console.log(document.querySelectorAll('.text > p u')); // NodeList

var p = document.getElementById('paragraph1');
p.style.backgroundColor = "red";
p.style.borderBottom = "1px solid blue";
document.querySelectorAll('.text > p u').forEach(function (el) {
  el.style.fontWeight = "bold";
  el.nextSibling.textContent = el.nextSibling.textContent + " modifié";
});
```

Bienvenue

- [Accueil](#)
- [Contact](#)

Titre : Contenu

Titre : Contenu

Bienvenue

- [Accueil](#)
- [Contact](#)

Titre : Contenu modifié

Titre : Contenu modifié

Manipulation de noeuds

L'objet `Node` retourné par les fonctions de sélection permet les opérations suivantes :

```
// On crée un nouveau noeud
var new_p = document.createElement('p');
// On lui insère du texte (ou autres opérations)
new_p.textContent = "Nouveau paragraphe";
// On sélectionne un élément
var p = document.getElementById('paragraph1');
// On lui ajoute le nouvel élément comme voisin
// (ajout d'un nouvel enfant au parent)
p.parentElement.appendChild(new_p);
// Et on le supprime
p.parentElement.removeChild(new_p);
// On sélectionne un autre élément
var header = document.getElementsByTagName('header')[0],
    nav = header.lastElementChild;
// Et on insère directement le nouveau paragraphe juste avant nav
header.insertBefore(new_p, nav);
// Puis on le supprime
header.removeChild(new_p);
```

Exécution de Javascript dans un document HTML

Il est possible de déclencher l'exécution d'un code Javascript depuis du HTML pour réagir aux actions de l'utilisateur :

```
<!DOCTYPE html>
<html>

<body>
  <button type="button" id="boutonDemanderNom" onclick="demanderNom()">Qui est-tu ?</button>
  <button type="button" id="boutonDireBonjour" style="visibility:hidden;">Bonjour !</button>
  <script type="text/javascript">
    function direBonjour(nom) {
      alert("Bonjour " + nom + " !");
    }

    function demanderNom() {
      var nom = prompt("Quel est ton nom ?");
      if (nom != "") {
        document.getElementById("boutonDireBonjour").style.visibility = "visible";
        document.getElementById("boutonDireBonjour").addEventListener('click', function () {
          direBonjour(nom);
        })
      }
    }
  </script>
</body>

</html>
```

Exercice 5 :

Développer un chronomètre
avec 3 boutons : Start, Stop et Lap

- Le temps écoulé doit être mis à jour toutes les 50 ms
- A chaque pression du bouton Lap, on enregistre le temps et on l'affiche dans une liste (ou)
- On peut rajouter un bouton Reset pour tout réinitialiser

00:31.947



Laps :

1. 00:01.340 (00:01.340)
2. 00:04.185 (00:02.845)
3. 00:08.550 (00:04.365)
4. 00:14.047 (00:05.497)
5. 00:15.915 (00:01.868)



VII. Programmation événementielle avec jQuery



Programmation événementielle

- Chaque action faite par l'utilisateur sur la page peut être écoutée :
 - Déplacement de la souris : `mousemove`, `mouseenter`, `mouseout`, `mouseover`
 - Clics de souris : `click`, `dblclick`, `contextmenu`, `mousedown`, `mouseup`
 - Pressions du clavier : `keypress`, `keydown`, `keyup`
 - Formulaire : `focus`, `blur`, `change`, `select`, `submit`

```
<button type="button" id="boutonDemanderNom" onclick="demanderNom()">Qui est-tu ?</button>
```

- Il est aussi possible d'écouter certaines actions du navigateur et de la page :
 - Défilement de la page : `scroll`
 - Redimensionnement du navigateur : `resize`
 - Chargement du DOM : `ready`

jQuery : Présentation


jQuery est une librairie de fonctions, servant plusieurs utilités :

- Parcours du DOM et sélection d'éléments
- Modification, création, suppression d'éléments
- Modifications de style et animations
- Ecoute d'évènements
- Requêtes AJAX

Malgré son ancienneté (2006), elle reste l'une des librairies JS les plus populaires, car facile à utiliser, et prérequis de beaucoup de librairies et framework "récents" (Bootstrap, Semantic UI, EmberJS, ..). Elle permet d'ajouter du "dynamisme" sans passer par une solution front complète du type Angular ou React.

jQuery : Parcours du DOM

- jQuery est “exposé” via deux variables globales identiques : **jQuery** et **\$**
- Cette variable est une fonction avec comme paramètre un sélecteur similaire à ceux contenus dans un fichier CSS pour sélectionner un ou plusieurs **HTMLElement** :



```
$('p#paragraphe1') // <p>  
$('section.text') // [<section>, <section>]  
$('body') // <body>
```

- On peut également passer un **HTMLElement** natif à la fonction \$, qui retournera un **HTMLElement** enrichi avec les méthodes de jQuery

jQuery : Parcours du DOM

- Une fois que l'on a sélectionné un élément, jQuery met à disposition des fonction pour se déplacer dans l'arborescence :
 - `$(...).children(<selecteur?>)` : Récupère les **enfants** (*fac: filtrés par un sélecteur*)
 - `$(...).find(<selecteur>)` : Récupère les **descendants** correspondant au sélecteur
 - `$(...).parent()` : Récupère le **parent**
 - `$(...).parents(<selecteur?>)` : Récupère les **ancêtres** (*fac: filtrés par un sélecteur*)
 - `$(...).siblings(<selecteur?>)` : Récupère les **voisins** (*fac: filtrés par un sélecteur*)
- Chacune de ces fonctions renvoie un ensemble d'éléments, comme le sélecteur standard de jQuery
- **Attention** : L'ensemble obtenu correspond à la fusion des résultats des `appels` sur chaque élément du sélecteur parent. Les méthodes pouvant être chaînées, l'ensemble final peut être très volumineux.
- La fonction `$(...).filter(<selector>)` permet de filtrer un ensemble existant

jQuery : Gestion des éléments

- Une fois un ensemble trouvé, on peut appliquer les opérations suivantes :
 - `$(...).html(<value>)` : Modification du **innerHTML** de l'élément (**inclus la descendance**)
 - `$(...).attr(<attribute>,<value>)` : Modification d'un attribut
 - `$(...).val(<value>)` : Modifie l'attribut **value** (*ex: champs de formulaire*)
 - `$(...).text(<value>)` : Modifie le contenu texte de l'élément
- Chaque fonction appelée sans **value** permettra de récupérer la valeur actuelle
- jQuery permet de créer des nouveaux éléments très simplement, par l'une de ces 3 méthodes (exemple avec une `<div>`) :
 - `$ (document.createElement('div'))`
 - `$("<div></div>")`
 - `$("<div>")`

jQuery : Gestion des éléments

- On peut ensuite insérer ce nouvel élément dans le DOM avec :
 - `$(<parent>).append(<el>) / $(<el>).appendTo(<parent>)` : Ajout en dernier enfant
 - `$(<parent>).prepend(<el>) / $(<el>).prependTo(<parent>)` : Ajout en premier enfant
 - `$(<selecteur>).before(<el>)` : Ajout avant un élément
 - `$(<selecteur>).after(<el>)` : Ajout après un élément
 - `$(<selecteur>).replaceWith(<el>)` : Remplace chaque élément par le nouveau
 - `$(<selecteur>).wrapWith(<el>)` : Encadre chaque élément (<el> devient son parent)
- On pourra aussi retirer des éléments avec :
 - `$(<selecteur>).remove()` : Supprime chaque élément sélectionné
 - `$(<selecteur>).empty()` : Supprime les descendants du noeud

jQuery : Gestion du CSS

- Le CSS d'un HTML Element peut être géré via la manipulation de classes :
 - `$ (...).hasClass (<cls>)` : Teste la présence d'une classe
 - `$ (...).addClass (<cls>)` : Ajout de classe
 - `$ (...).removeClass (<cls>)` : Suppression de classe
 - `$ (...).toggleClass (<cls>)` : Inverse la présence d'une classe (l'ajoute si elle est absente et inversement)
- Ou directement en gérant les propriétés de l'élément dans le DOM
 - `$ (...).css (<property>)` : Récupère la valeur d'une propriété CSS (qu'elle soit inscrite dans le DOM ou dans un fichier)
 - `$ (...).css (<property>, <value>)` : Modifie la valeur d'une propriété CSS (écrit dans le DOM, ce qui prend la priorité sur le CSS lu depuis un fichier)

jQuery : Ecoute d'événements

- jQuery permet de surcharger les écouteurs d'événements du DOM avec sa fonction `on` : `$(<selecteur>).on(<evenement>, <action>);`

```
// Avec une fonction nommée
$('button.closeButton').on('click', close);
// Avec une fonction anonyme
$('button.okButton').on('click', function(e){
    obj.save();
});
```

- Il faut également parfois désactiver les écouteurs :

```
// Suppression d'un seul écouteur si on a la référence à la fonction
$('button.closeButton').off('click', close);
// Suppression de tous les écouteurs sur les éléments sélectionnés
$('button.okButton').off('click');
```

jQuery : Class Event

- jQuery (tout comme le DOM) passe un objet **Event** en premier paramètre de la fonction lorsqu'un événement est déclenché. On le symbolise souvent par la variable `e` (ou `evt`). Il contient énormément d'informations, par exemple dans le cas d'un clic de souris :
 - Le bouton de souris utilisé
 - Les coordonnées du clic
 - L'élément visé par le clic (attention si l'élément écouté a des descendants !)
 - Les éventuels modificateurs (Control, Alt, Shift, Meta)
- L'élément visé par le clic peut être récupéré dans la variable **this** à l'intérieur de la fonction
- Il est possible d'annuler l'effet originel de l'événement en utilisant la méthode **preventDefault** et/ou **return false**

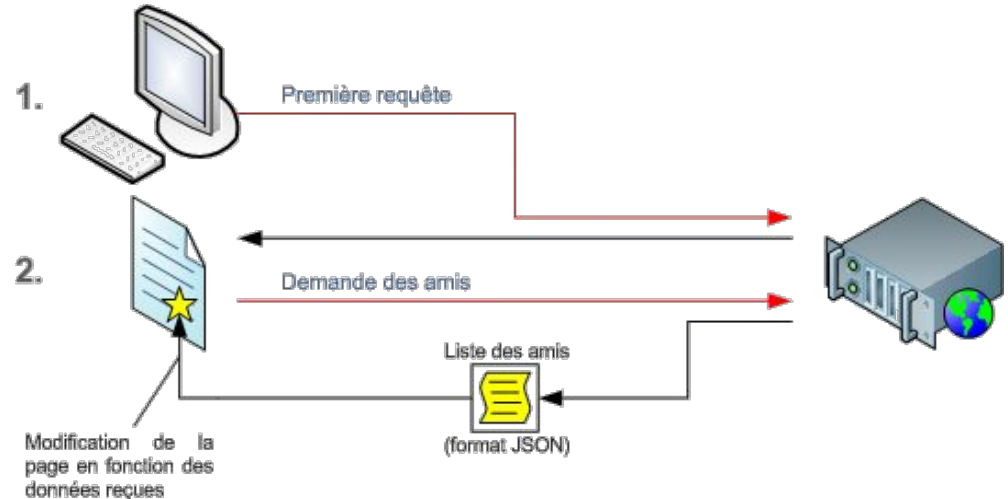
jQuery : Écouteurs d'événements et Scopes

- Puisque la variable **this** est affectée à l'intérieur d'un écouteur d'événement, il faut ruser si l'on souhaite profiter de la permissivité de Javascript au sujet de la visibilité des variables dans les différents scopes d'exécution :

```
class User {  
  constructor(email) {  
    this.email = email;  
  }  
  
  setupEvents() {  
    var me = this;  
    $('button#sendButton').on('click', function () {  
      // "this" contient le HTMLInputElement du bouton  
      // "me" est l'instance de la class User  
      mail(me.email, "Test");  
    });  
  }  
}
```

jQuery : Requêtes AJAX

- AJAX = Asynchronous Javascript And XML
- jQuery a révolutionné le développement Web, en surchargeant l'objet XHR (XML HTTP Request) natif, avec des méthodes simples à utiliser. Il n'est désormais plus nécessaire de recharger l'intégralité des pages pour changer un seul élément :



jQuery : Requêtes AJAX

- Il met pour cela à disposition plusieurs méthodes très similaires :
 - `$.get(<url>, <params>)` : Envoie une requête GET
 - `$.post(<url>, <params>)` : Envoie une requête POST
 - `$.ajax(<url>, <params>)` : Envoie une requête dont le verbe est spécifié dans les params
 - `$.getJSON(<url>, <params>)` : Envoi d'une requête GET dont la réponse sera du JSON
- Dans l'objet params, on ira spécifier les détails, comme par exemple :
 - Les paramètres de requête (formulaire, filtres, tris, ...) et le type de réponse
 - Le timeout (durée d'attente avant échec)
 - Les callbacks : fonctions qui seront exécutées selon l'état de la réponse :
 - **success** et **error**
 - **complete** : La requête est terminée, peu importe si il y a eu une erreur ou non
 - ...
 - ...

Exercice 6 : Jeu du Pendu

Développer un jeu de pendu (HTML/CSS/JS) :

- Choix aléatoire d'un mot depuis un tableau
- Affichage de la première et dernière lettre, et des underscores entre
- 26 boutons de lettres, qui sont désactivés après avoir été choisis
- 6 checkbox "readonly" qui sont cochées à chaque erreur
- Au bout de 6 erreurs, c'est perdu ; si le mot est trouvé, c'est gagné
- Permettre au joueur de recommencer une nouvelle partie
- Bonus : Permettre le choix des lettres au clavier

Exercice 6 : Jeu du Pendu

Le Pendu

I M _ _ E _ _ I O N

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

☒ Erreur 1
☒ Erreur 2
☒ Erreur 3

<https://code.jquery.com>

pour récupérer le CDN de jQuery

<http://www.gerenimot.fr/>

pour des mots aléatoires
(retirer les accents)

Le Pendu

P A _ I _ _ E _ I E

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

☒ Erreur 1
☒ Erreur 2
☒ Erreur 3
☒ Erreur 4
☒ Erreur 5
☒ Erreur 6

PERDU !

Le Pendu

A N C E T R E

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

☒ Erreur 1
☒ Erreur 2
☒ Erreur 3
☒ Erreur 4

GAGNÉ !



VIII. Fin du Sprint



Fonctions utiles pour le projet

```
/**
 * Efface l'écran de la console
 * @return void
 */
function cls(){
    if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN'){
        system('cls');
    } else {
        system('clear');
    }
}
```

```
/**
 * Affiche la chaîne suivie d'un retour à la ligne
 * @param $str Chaîne à afficher
 * @return void
 */
function println(string $str){
    echo $str . PHP_EOL;
}
```

IX. Lab' Day