# Premiers contacts avec JavaScript

On a vu que l’on peut créer des formulaires pour transmettre des données. On peut avoir besoin de s’assurer du bon format des données qui seront envoyées. L’envoie de paramètres se fait uniquement par une requête HTTP, et donc implique une communication en réseau avec le site distant et le serveur doit recalculer entièrement la page de destination. L’utilisateur voit donc une nouvelle page se charger. L’interactivité des sites Web est donc limitée et contrainte par un cycle : remplissage d’un formulaire, soumission de ce formulaire, calcule plus affichage de la page résultante.

Le constat est alors simple. Si le fait de calculer *côté serveur* implique des aller-retours via le réseau, le formatage de paramètres peut se passer *côté client* ! C’est ce que permet le langage JavaScript, dont un interpréteur est intégré dans tous les navigateurs grand public.

## Un peu d’histoire

Le langage JavaScript a été créé dans les années 1995 par Brendan Eich, un informaticien américain alors employé de l’éditeur de logiciels Netscape Communications. L’un des produits phare de cette société était le navigateur Web Netscape Navigator, qui est le premier à intégrer JavaScript. À l’époque, le système Windows de Microsoft et son navigateur Internet Explorer, intégré au système, sont omniprésents. Netscape Navigator ne peut pas survivre en tant que logiciel commercial. Son code est ouvert et il sert de base au navigateur Firefox, toujours en développement actuellement.

Dés 1996, l’adoption de JavaScript, tant par les utilisateurs de Netscape Navigator que par ceux d’Internet Explorer (Microsoft ayant intégré à Internet Explorer leur propre version de JavaScript) en fait un langage incontournable pour la programmation Web côté client.

## Premiers pas

**À faire vous-même n°1 :**

Écrire le code suivant dans un fichier HTML et l’ouvrir avec le navigateur Firefox par exemple. Il n’est pas nécessaire d’avoir de serveur Web.

<html>

<head><title>Age</title>

<script type="text/javascript">

let compteur = 0 ;

function suivant() {

compteur = compteur + 1 ;

document.getElementById("valeur").innerHTML = compteur;

}

</script>

</head>

<body>

<button onclick="suivant();">Clic!</button>

<span id="valeur">0</span>

</body>

</html>

Si on ignore dans un premier temps la balise <script> située dans l’entête, on constate que le corps du document contient deux éléments. Un bouton dont le texte est « Clic ! » et un élément <span> dont le texte est 0. On remarque que le bouton est en dehors de tout formulaire. L’élément <span> possède un id. Le bouton possède un attribut onclick dont la valeur est une chaîne de caractères contenant un appel de fonction. Lorsque l’on clique sur ce bouton, la fonction est appelée. Nous sommes dans le cadre de la programmation événementielle d’interfaces interactives.

Nous allons expliquer le contenu de la balise <script> du document HTML précédent. Le script commence par définir une variable compteur (« let » permet de déclarer des variables dont la portée est limitée à celle du bloc dans lequel elles sont déclarées), initialisée à 0. Il définit ensuite une fonction suivant(). Lorsqu’elle est appelée, cette dernière commence par incrémenter la variable compteur. L’instruction suivante, récupère un objet correspondant à l’élément de la page dont l’id vaut valeur. Cela correspond à l’élément <span id="valeur"> se trouvant après le bouton. Une fois cet élément récupéré, il est possible d'écraser le contenu textuel (initialement le 0) par une nouvelle valeur. Toutes ces modifications sont reflétées directement dans l'affichage de la page, où l'on peut voir directement la valeur du compteur augmenter **sans que la page ne se recharge**. Par contre, si on recharge manuellement la page (par exemple avec le bouton « recharger » de navigateur) alors le programme recommence du début.

**À faire vous-même n°2 :** Modification *live* d’une page web.

Créer un fichier test.html sur Visual Studio Code avec le code suivant :

|  |
| --- |
| <!DOCTYPE html>  <html lang="fr">  <head>  <meta charset="UTF-8">  <title>Test</title>  </head>  <body>  <div id="mondiv">Hello</div>  </body>  </html> |

Ouvrir ce fichier avec le navigateur Firefox. Ouvrir la *console JavaScript* en utilisant le menu « outils » puis « Développement Web ». Écrire les instructions suivantes dans la console et constater que l’affichage de la page se modifie automatiquement. Il est toujours possible de recharger la page pour se retrouver dans l’état initial (*i.e.* le fichier stocké sur le disque n’est pas modifié).

* Changement de la couleur de fond de la page :

document.body.style.background = "red";

* Ré-initialisation à la couleur par défaut :

document.body.style.background = "";

* Récupération d’un élément. Cette instruction doit être faite une fois pour que d soit définie, avant de tester la suite :

let d = document.getElementById("mondiv");

* Changement du style de l’élément d :

d.style.color = "blue";

d.style.fontWeight = "bold";

d.style.border = "1pt dashed green";

* Changement du contenu de l’élément d :

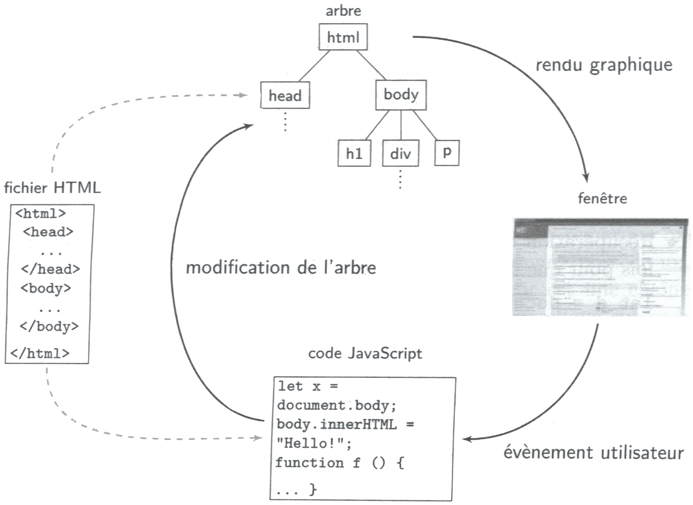
d.innerHTML = "Salut, <i>ça va bien ?</i>";

Au travers de ces différents exemples, on constate les changements qui s’effectuent sur la page sans que celle-ci ne soit rechargée à l’aide de lignes de code écrites en JavaScript.

# Évaluation du code JavaScript

## Une structure de données pour HTML

Comme on l’a vu dans les deux exemples précédents, l'un des principaux attraits de l'interpréteur JavaScript d'un navigateur est la possibilité de modifier dynamiquement et en temps réel le rendu d'une page Web. Le schéma ci-dessous décrit succinctement le fonctionnement d'un navigateur Web. Le navigateur commence par télécharger le fichier HTML correspondant à l'URL visée. Une fois récupérée, la page est transformée en *une structure de données d'arbre*. Une fois l'arbre construit, ses nœuds (les éléments du document HTML) sont parcourus, les règles CSS sont appliquées et le rendu graphique de la page est effectué. Dans le même temps, les scripts JavaScript présent dans la page sont exécutés. L'une de leur activité principale est la définition de *gestionnaires d'événements* que l'on verra dans la suite de ce cours. Ces gestionnaires d'événements peuvent modifier librement l'arbre représentant le document. Ajouter, supprimer ou modifier des nœuds dans l'arbre correspond à ajouter, supprimer ou modifier les balises correspondantes dans le document (encore une fois, le fichier original qu'il soit stocké localement ou sur le serveur Web n'est pas modifié). Les attributs des balises peuvent aussi être modifiés. Parmi ces attributs, la modification de l'attribut class permet de changer le rendu graphique de l'élément. Ces changements sont immédiatement répercutés sur l'affichage.



## Programmation évènementielle en JavaScript

Comme nous l'avons vu dans notre exemple introductif, on peut ajouter à un bouton un attribut onclick contenant du code JavaScript. Ce dernier sera exécuté si l'utilisateur clique sur le bouton. De la même manière, on pourrait ajouter des attributs onkeypress (pression sur une touche du clavier), onmouseover (la souris survole l'élément) ou tout autre attribut d'événement parmi les dizaines d'événements possibles. De plus, ces attributs ne sont pas spécifiques aux boutons. N'importe quel élément HTML peut être décoré d'un tel attribut. Par convention, le nom des attributs commence par on (pour l'anglais « en cas de »).

Cependant, même si cette technique fonctionne et reste utilisée de temps à autre, elle tend fortement à disparaître car elle possède de nombreux inconvénients. Premièrement, elle souffre du défaut de mélanger la structure du document HTML avec un concept étranger (la présentation graphique dans le cas de style, du code JavaScript dans le cas des attributs on…). Deuxièmement, cette approche rend le débuggage et la maintenance de code difficile, le code JavaScript est éparpillé dans les attributs des balises HTML. Qui plus est, ce dernier est écrit sous forme de chaîne de caractères, ce qui rend la lecture compliquée.

Les bonnes pratiques actuelles, connu sous le nom de « JavaScript discret » (ou en anglais *unobtrusive JavaScript*) prônent une séparation stricte du code JavaScript, du HTML et du CSS. Pour écrire le code JavaScript dans un fichier séparé, on précise la source dans la balise <script>, c’est-à-dire l’adresse du fichier contenant le code JavaScript, avec l’attribut src. Les balises <script> et </script> se situent dans l’entête entre les balises <head> et </head>.

# Jeu : Deviner un nombre

**Cahier des charges :**

Nous allons créer ensemble un jeu simple de devinette de nombre. Le jeu choisit aléatoirement un nombre entre 1 et 100, puis il met le joueur au défi de le deviner en 10 tentatives maxi. À chaque tour, le joueur doit être informé s'il a deviné ou non le bon nombre — si ce n'est pas le cas, le jeu lui indique si son estimation est trop basse ou trop élevée. Le jeu doit également rappeler au joueur les nombres déjà proposés. Le jeu se termine quand le joueur a deviné le nombre mystère, ou s'il a épuisé ses 10 chances. À la fin du jeu, le joueur a la possibilité de débuter une nouvelle partie.

**À faire vous-même n°1 :**

Familiarisez-vous avec ce jeu avant de continuer en ouvrant le fichier jeu.html dans le commun de la classe avec un navigateur tel Firefox.

**À faire vous-même n°2 :**

La première chose à faire, c'est de le décomposer en tâches simples et codables comme le ferait un programmeur :

|  |
| --- |
|  |

Voyons maintenant comment nous pouvons transformer ces étapes en code. Nous allons développer cet exemple et explorer les fonctionnalités JavaScript au fur et à mesure.

**À faire vous-même n°3 :**

*Nous allons avoir besoin de trois fichiers :*

1. Un fichier HTML qui nous permettra de mettre en forme notre page. Créer un dossier jeu dans vos documents puis copier le fichier index.html qui se trouve dans le commun que vous devrez coller dans votre dossier jeu.
2. Un fichier CSS qui nous permettra de styliser la page. Créer un dossier css dans le dossier jeu puis c*opier le* fichier style.css *qui se trouve dans le commun de la classe* dans le dossier css.
3. Un fichier JavaScript où l’on notera nos scripts. Créer un dossier js dans le dossier jeu puis créer un fichier script.js dans le dossier js.

Il faut toujours penser à organiser son arborescence tel que l’on le ferait pour un plus grand projet.

**Configuration initiale**

**À faire vous-même n°4 :**

Pour commencer, ouvrez le fichier *index.html* dans Visual Studio Code et dans un navigateur web :

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Jeu : deviner un nombre</title>

</head>

<body>

<h1>Deviner un nombre</h1>

<p>Nous avons généré un nombre aléatoire entre 1 et 100, tentez de le deviner en 10 tours maximum. Pour chaque tentative, nous vous dirons si votre estimation est trop ou pas assez élevée.</p>

<div class="form">

<label for="guessField">Entrer un nombre : </label>

<input type="text" id="guessField" class="guessField">

<input type="submit" value="Vérifier" class="guessSubmit">

</div>

<div class="resultParas">

<p class="guesses"></p>

<p class="lastResult"></p>

<p class="lowOrHi"></p>

</div>

</body>

</html>

Pour l'instant, on ne voit qu'un titre, un paragraphe d'instructions, un formulaire pour entrer une estimation, mais le formulaire est pour l'instant inactif et trois paragraphes vides. C’est dans ces paragraphes que l’on fera apparaître des informations sur le jeu avec le JavaScript.

**À faire vous-même n°5 :**

Pour le moment, insérer les deux lignes suivantes entre les balises <head> et </head> de votre fichier HTML :

*<link rel="stylesheet" href="css/jeu.css">*

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

La première ligne permet d’indiquer que la page utilise une feuille de style et où se trouve le fichier correspondant.

La deuxième ligne indique que la page HTML utilise du JavaScript et où se trouve le fichier correspondant.

**Ajouter des variables pour stocker des données**

**À faire vous-même n°6 :**

Tout d'abord, ajoutez les lignes suivantes dans le fichier JavaScript, script.js :

let randomNumber = Math.floor(Math.random() \* 100) + 1;

let guesses = document.querySelector('.guesses');

let lastResult = document.querySelector('.lastResult');

let lowOrHi = document.querySelector('.lowOrHi');

let guessSubmit = document.querySelector('.guessSubmit');

let guessField = document.querySelector('.guessField');

let guessCount = 1;

let resetButton;

Cette partie de code définit les variables nécessaires au stockage des données que notre programme utilisera. Les variables sont essentiellement des conteneurs de valeurs (tels que des nombres ou des chaînes de texte). Une variable locale se crée avec le mot-clé*let* suivi du nom de la variable. Vous pouvez ensuite attribuer une valeur à la variable avec le signe égal (*=*) suivi de la valeur que vous voulez lui donner.

Dans notre exemple :

* La première variable — randomNumber — reçoit le nombre aléatoire entre 1 et 100, calculé en utilisant un algorithme mathématique. Les trois variables suivantes sont chacune faite pour stocker une référence aux paragraphes de résultats dans le HTML. La méthode querySelector() de l'interface [Document](https://developer.mozilla.org/fr/docs/Web/API/Document) retourne le premier [Element](https://developer.mozilla.org/fr/docs/Web/API/Element" \o "Element est la classe de base la plus générale à partir de laquelle tous les objets d'un Document héritent. Il n'a que des méthodes et des propriétés communes à tous les types d'éléments. Les classes plus spécifiques héritent d'Element. Par exemple, l'interface HTMLElement est l'interface de base pour les éléments HTML, tandis que l'interface SVGElement est la base de tous les éléments SVG. La plupart des fonctionnalités sont spécifiées plus bas dans la hiérarchie des classes.) dans le document correspondant au sélecteur - ou groupe de sélecteurs spécifié(s), ou null si aucune correspondance n'est trouvée.

Elles sont utilisées pour insérer plus tard des valeurs dans les paragraphes suivants de la page HTML :

<p class="guesses"></p>

<p class="lastResult"></p>

<p class="lowOrHi"></p>

* Les deux variables —  guessSubmit et  guessField  — stockent des références au champ de saisie du formulaire et au bouton de soumission ; elles sont utilisées pour écouter l'envoi de la proposition par le formulaire de la page HTML :

<input type="text" id="guessField" class="guessField">

<input type="submit" value="Vérifier" class="guessSubmit">

Nos deux dernières variables stockent un nombre de suppositions qui vaut initialement 1 (utilisées pour garder une trace du nombre de suppositions que le joueur a fait) et une référence à un bouton de réinitialisation qui n'existe pas encore.

**Fonctions**

**À faire vous-même n°7 :**

Ajoutez maintenant ce qui suit dans votre code JavaScript :

function checkGuess(){

let userGuess = Number(guessField.value);

if (guessCount === 1) {

guesses.textContent = 'Propositions précédentes : ';

}

guesses.textContent += userGuess + ' ';

if (userGuess === randomNumber) {

lastResult.textContent = 'Bravo, vous avez trouvé le nombre !';

lastResult.style.backgroundColor = 'green';

lowOrHi.textContent = '';

setGameOver();

} else if (guessCount === 10) {

lastResult.textContent = '!!! PERDU !!!';

setGameOver();

} else {

lastResult.textContent = 'Faux !';

lastResult.style.backgroundColor = 'red';

if (userGuess < randomNumber) {

lowOrHi.textContent = 'Le nombre saisi est trop petit !';

} else if (userGuess > randomNumber) {

lowOrHi.textContent = 'Le nombre saisi est trop grand !';

}

}

guessCount = guessCount + 1;

guessField.value = '';

guessField.focus();

}

Pas mal de code — ouf ! Passons en revue chaque section et expliquons ce qu'elle fait.

* La première ligne de la fonction (ligne 2) déclare une variable nommée userGuess et définit sa valeur par celle qui vient d'être saisie dans le champ de texte. Nous faisons passer aussi cette valeur par la méthode Number(), juste pour nous assurer que la valeur stockée dans userGuess est bien un nombre.
* Ensuite, nous passons au bloc de code conditionnel (lignes 3-5). Le test vérifie si la variable guessCount est égale à 1 (c'est-à-dire s'il s'agit de la première supposition du joueur). Si c'est le cas, nous faisons en sorte que le texte affiché soit « Propositions précédentes : ». Sinon, nous ne le faisons pas.
* La ligne 6 ajoute la valeur courante userGuess à la fin du paragraphe guesses, plus un espace vide de sorte qu'il y aura un espace entre chaque supposition faite.
* Le bloc suivant (lignes 8-24) effectue quelques vérifications :
  + Le premier if(){ } vérifie si la supposition de l'utilisateur est égale au nombre aléatoire randomNumber situé en haut de notre code JavaScript. Si c'est le cas, le joueur a deviné correctement et a gagné le jeu, nous affichons donc un message de félicitations d'une belle couleur verte au joueur, effaçons le contenu de la boîte d'information sur la position de l'estimation et exécutons une fonction appelée setGameOver(), dont nous reparlerons plus tard.
  + Ensuite, nous chaînons un autre test à la fin du précédent avec une structure else if(){ }. Cette structure vérifie si l'utilisateur a épuisé toutes ses tentatives. Si c'est le cas, le programme fait la même chose que

dans le bloc précédent, mais avec un message de fin de partie au lieu d'un message de félicitations.

* + Le dernier bloc chaîné à la fin de ce code (else { }) contient du code qui n'est exécuté que si aucun des deux autres tests n'a renvoyé **vrai**  (c'est-à-dire que le joueur n'a pas deviné juste, mais qu'il lui reste des possibilité de supposition). Dans ce cas, nous lui disons que sa supposition est mauvaise, puis nous effectuons un autre test conditionnel pour vérifier si elle est supérieure ou inférieure à la valeur exacte et affichons un autre message approprié pour indiquer si sa supposition est trop forte ou trop faible.
* Les trois dernières lignes de la fonction (ligne 26-28) préparent à une nouvelle proposition. Nous ajoutons 1 à la variable guessCount qui décompte les tours (++ est une opération d'incrémentation — ajout de 1), puis effaçons le champ texte du formulaire et lui redonnons le focus, pour être prêt pour la saisie suivante.

**Évènements**

À ce stade, nous avons bien implémenté la fonction checkGuess(), mais elle ne s'exécutera pas parce que nous ne l'avons pas encore appelé.  
Idéalement, nous voulons l'appeler lorsque le bouton guessSubmit est cliqué, et pour ce faire, nous devons utiliser un événement. Les événements sont des actions qui se produisent dans le navigateur, comme le clic sur un bouton, le chargement d'une page ou la lecture d'une vidéo, en réponse à quoi nous pouvons exécuter des blocs de code. On ajoute un attribut onclick dont la valeur est une chaîne de caractères contenant un appel de fonction. Lorsque l’on clique sur ce bouton, la fonction checkGuess() est appelée.

**À faire vous-même n°8 :**

Remplacer dans le fichier HTML le bouton guessSubmit  :

<input type="submit" value="Vérifier" class="guessSubmit" onclick="checkGuess();">

Essayez d'enregistrer et d'actualiser votre code, votre exemple devrait désormais fonctionner, jusqu'à un certain point. Maintenant, le seul problème est que si vous devinez la bonne réponse ou si vous n'avez plus de tours à jouer, le jeu "va se casser" parce que nous n'avons pas encore implémenté la fonction setGameOver() dont le rôle est de terminer proprement le jeu. Ajoutons maintenant le code manquant pour compléter notre exemple.

**Finir les fonctionnalités du jeu**

**À faire vous-même n°9 :**

Pour définir la fonction setGameOver() à la fin de notre programme, ajoutez le code ci-dessous tout en bas :

function setGameOver() {

guessField.disabled = true;

guessSubmit.disabled = true;

resetButton = document.createElement('button');

resetButton.textContent = 'Démarrer une nouvelle partie';

document.body.appendChild(resetButton);

resetButton.addEventListener('click', resetGame);

}

* Les deux premières lignes désactivent l'entrée de texte et le bouton en définissant leurs propriétés désactivées à true.  Ceci est nécessaire, car si nous ne le faisons pas, l'utilisateur pourrait soumettre plus de propositions après la fin du jeu, ce qui gâcherait les choses.
* Les trois lignes suivantes génèrent un nouvel [<button>](https://developer.mozilla.org/fr/docs/Web/HTML/Element/button) élément, avec le libellé "Démarrer une nouvelle partie" et l'ajoute au bas du HTML existant.
* La dernière ligne définit **un écouteur d'événement** sur ce nouveau bouton : un clique sur le bouton déclenchera un appel de la fonction resetGame(). Ici, nous ajoutons un écouteur d'événement au bouton resetButton. C'est une méthode qui prend deux valeurs d'entrée (appelées arguments) - le type d'événement que nous écoutons (dans ce cas, click) qui est une chaîne de caractères, et le code que nous voulons exécuter quand l'événement se produit (dans ce cas, la fonction resetGame()  — notez que nous n'avons pas besoin de spécifier les parenthèses lors de l'écriture dans addEventListener()).

**À faire vous-même n°10 :**

R*este à définir cette fonction ! Ajoutez le code suivant, tout en bas de votre JavaScript :*

function resetGame() {

guessCount = 1;

let resetParas = document.querySelectorAll('.resultParas p');

for (let i = 0 ; i < resetParas.length ; i++) {

resetParas[i].textContent = '';

}

resetButton.parentNode.removeChild(resetButton);

guessField.disabled = false;

guessSubmit.disabled = false;

guessField.value = '';

guessField.focus();

lastResult.style.backgroundColor = 'white';

randomNumber = Math.floor(Math.random() \* 100) + 1;

}

Ce bloc de code assez long réinitialise complètement les paramètres du jeu (le joueur pourra commencer une nouvelle partie). Il permet de :

* Remettre le compteur guessCount à 1.
* Effacer tous les paragraphes d'information.
* Supprimer le bouton de réinitialisation de notre code.
* Activer les éléments de formulaire, vide et met au point le champ de texte, prêt à entrer une nouvelle proposition.
* Supprimer la couleur d'arrière-plan du paragraphe lastResult.
* Générer un nouveau nombre aléatoire afin que vous ne deviniez plus le même nombre !

**À ce stade, vous devriez avoir un jeu (simple) entièrement fonctionnel — *félicitations !***