Henallux – DA/IA2 – Technologies web

Laboratoire 1

Introduction à JavaScript

Sommaire

[1. Informations générales sur le cours 2](#_Toc176510420)

[2. Prise en main de la console Firefox 6](#_Toc176510421)

[3. Une Première page HTML-dynamique 8](#_Toc176510422)

[4. révision des tables de multiplication 12](#_Toc176510423)

[5. compteur, boutons et liens 16](#_Toc176510424)

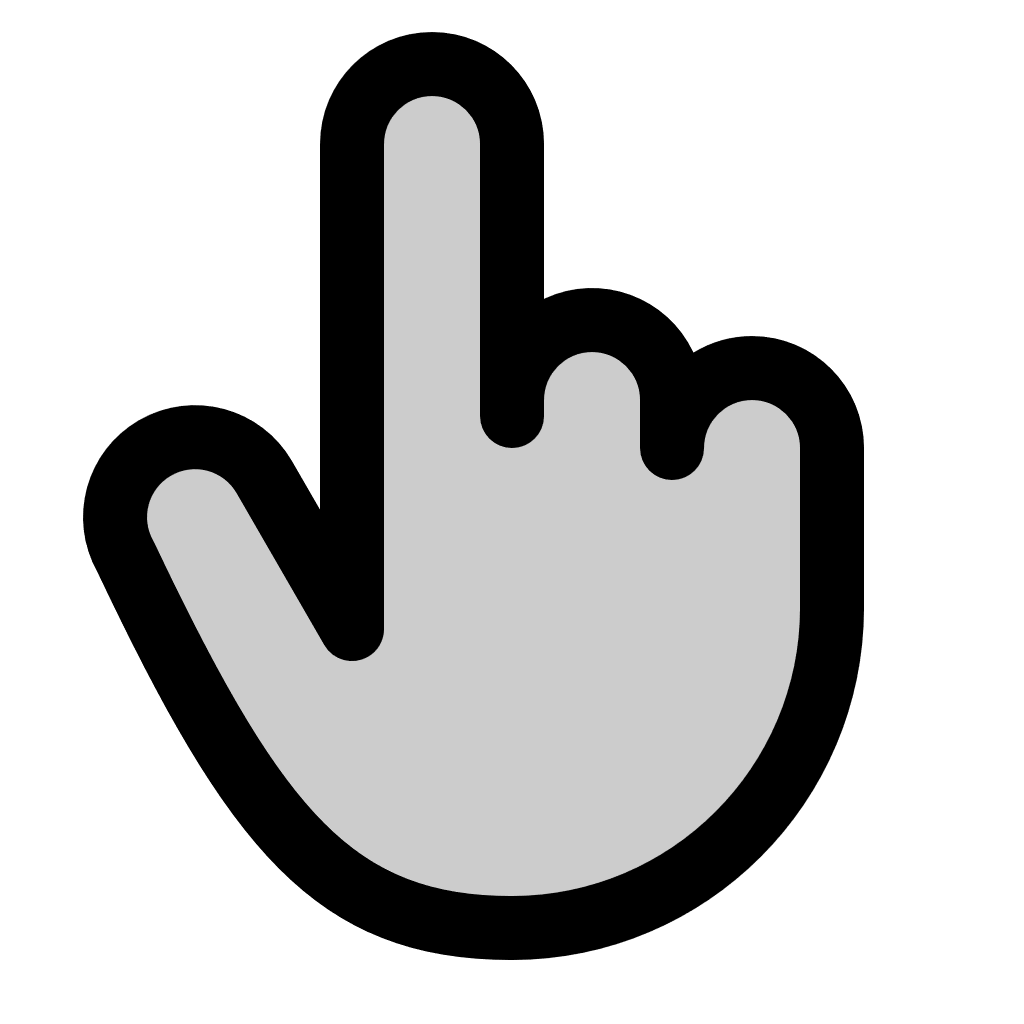
[6. Mini-tableur 18](#_Toc176510425)

# Informations générales sur le cours

## Ateliers, labos et points théoriques

Le cours de Technologies web (JavaScript) se décline en trois types de contenu :

* des ateliers qui ont pour objectif de vous faire découvrir la matière et de vous présenter des situations qui illustrent l’utilité des nouveaux concepts introduits (il s’agit généralement d’exercices dirigés prévus pour être réalisés avant le point théorique),
* des exposés théoriques pour faire le point sur les concepts théoriques et introduire les notions plus difficiles (vous profiterez bien mieux de ce qui y sera dit si vous avez réalisé les ateliers à l’avance, vu que vous aurez déjà une idée de quoi ça parle), et
* des laboratoires qui vous permettent d’appliquer la matière vue via des exercices pratiques et introduisent également certains points qui ne nécessitent pas forcément un exposé théorique dédié.

**Note importante**

Les exposés théoriques ne contiennent pas forcément *toute* la matière du  
cours : certains éléments sont présentés au sein des ateliers et laboratoires.

Tous les modules ne contiennent pas forcément un atelier, un exposé théorique et un laboratoire. Par exemple, certains sujets plus complexes commencent directement par un exposé théorique, alors que d’autres, plus liés à des pratiques à exercer, n’ont pas d’exposés théoriques.

## Horaire du cours

D’un point de vue logistique, les exposés théoriques sont les seules parties du cours qui correspondent à une date et à une heure fixe ; ces parties-là sont difficilement modulables.

Les ateliers et les laboratoires, au contraire, peuvent être réalisés pendant les séances prévues ou en-dehors de celles-ci : à vous de gérer votre temps selon l’arrangement qui vous convient le mieux.

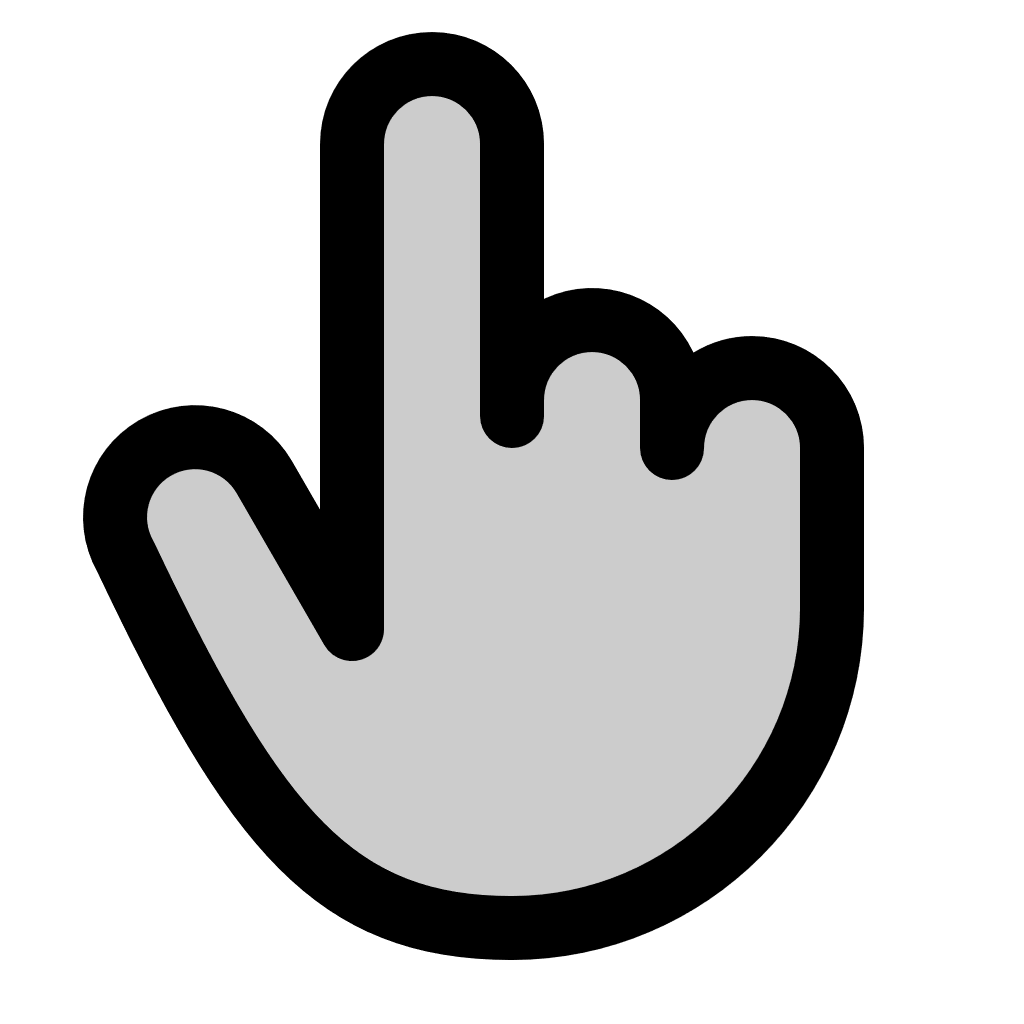
Vous pouvez considérer les séances d’ateliers et de labos comme des plages horaires où les enseignants sont disponibles pour répondre à vos questions et proposer des feedbacks sur votre code, mais c’est à vous de décider quand et comment vous travaillez sur ces parties du cours.

## Comment profiter au mieux des ateliers ?

* Gérez votre emploi du temps pour réaliser les ateliers avant les exposés théoriques correspondant.  
    
  Ça vous permettra d’avoir déjà un premier aperçu de la matière, de mieux comprendre le pourquoi et le comment des concepts abordés et aussi de bien plus facilement les intégrer et les retenir.
* Ne considérez pas les ateliers comme des tâches à réaliser aussi vite que possible. Prenez le temps de bien comprendre ce qu’on vous demande de faire et pourquoi (ne vous contentez pas d’exécuter bêtement les instructions données).
* Pendant la réalisation de l’atelier, prenez note de vos questions et des difficultés que vous rentrez… ça vous permettra d’en discuter après la séance théorique si elle n’a pas suffi à vous éclairer.
* Pendant la réalisation de l’atelier, mettez en évidence les points importants qui y sont introduits, ajoutez des notes sur les sujets qui y sont présentés. Cela facilitera vos révisions avant l’examen et cela vous permettra également de retrouver les informations plus facilement pendant celui-ci (qui se fait à cours ouvert).
* N’hésitez pas à comparer votre code et celui d’autres étudiants : c’est un bon exercice pour « rentrer dans le code de quelqu’un d’autre » et pour discuter des différentes manières de faire et de leurs avantages respectifs !

## Comment profiter au mieux des laboratoires ?

* Les laboratoires constituent normalement la dernière partie d’un module ; il n’y a pas vraiment de date-buttoir pour les terminer (contrairement aux ateliers qui sont suivis d’un point théorique).  
    
  Ceci dit, à nouveau, gérez votre emploi du temps pour ne pas prendre trop de retard : la matière s’accumule vite et les modules suivants réutilisent les notions vues précédemment (donc autant bien les avoir assimilées pour mieux profiter des ateliers qui suivent).
* Quand vous rédigez du code pour un laboratoire, gardez en tête toutes les notions de clean code, de réflexion et d’efficacité qui ont été abordées en 1re. Elles continuent de s’appliquer et, plus souvent vous prendrez l’habitude de les suivre, plus elles deviendront naturelles.
* Une fois votre code rédigé (et testé), prenez le temps de le relire à la recherche de voies d’amélioration : meilleure structure, écriture plus efficace, algorithme plus performant, utilisation de nouveaux concepts…
* Ne vous contentez pas de « ça marche ! » mais assurez-vous chaque fois de bien comprendre pourquoi. Comprendre pourquoi une erreur se produit est souvent une des parties les plus importantes de l’apprentissage !
* N’hésitez pas à comparer votre code et celui d’autres étudiants : c’est un bon exercice pour « rentrer dans le code de quelqu’un d’autre » et pour discuter des différentes manières de faire et de leurs avantages respectifs !
* N’hésitez pas à montrer à un enseignant un code (ou mieux, un bout de code) dont vous n’êtes pas certain ou sur lequel vous voudriez avoir un feedback. Il vaut aussi mieux envoyer des questions / du code à examiner de manière régulière plutôt que d’attendre d’avoir fini tout un laboratoire et d’envoyer plusieurs exercices en même temps !
* Comme pour les ateliers, prenez le temps de noter les éléments importants présentés dans les laboratoires au moment où vous les réalisez. Une mini table des matières reprenant les concepts décrits dans le document du laboratoire vous aidera à la fois pour vos révisions et pour l’examen !

**Note importante**

Une grande partie de ce cours dépend de votre organisation : c’est à vous  
de construire votre emploi du temps (sans doute sur base de l’horaire des  
points théoriques) pour vous assurer de bien profiter des ateliers et des laboratoires !  
  
Ce n’est pas forcément une tâche si facile que ça, et vous avez bien sûr le droit à l’erreur. Mais gardez un œil attentif pour vous rendre compte des problèmes (typiquement, si vous prenez beaucoup de retard) assez tôt pour pouvoir y remédier et en déduire quel mode de fonctionnement vous convient le mieux.

## Comment les laboratoires sont-ils organisés ?

Les laboratoires sont présentés sous la forme de listes d’exercices décomposés eux-mêmes en étapes. Au début de chaque exercice sont précisés les objectifs de celui-ci.

Pour gagner du temps (et éviter de vous en faire perdre), bon nombre d’exercices sont accompagnés de fichiers partiels ou de bouts de texte à copier/coller afin que vous puissiez vous concentrer sur les parties intéressantes plutôt que de créer à chaque fois tout à partir de zéro.

Pour certaines étapes, plus complexes ou nécessitant une réflexion plus approfondie, vous trouverez des indices (voire des solutions, complètes ou partielles) en fin de document. Pour profiter au mieux de ces exercices, il vaut mieux ne pas aller lire ces indices trop tôt : prenez tout d’abord le temps de bien réfléchir puis ne vous rendez en fin de document que lorsque vous êtes complètement bloqués ou que vous avez trouvé une réponse que vous désirez vérifier !

## De quoi ai-je besoin pour réaliser ces exercices ?

La plupart des exercices ne nécessite que deux logiciels :

* un éditeur de texte pour créer et modifier des fichiers (HTML, CSS ou JS), et
* un navigateur.

Les logiciels conseillés sont les suivants. Libre à vous d’en utiliser d’autres (ou d’autres environnements de travail) si vous le désirez mais soyez conscients que cela risque de vous faire passer à côté de certains apprentissages (et, qu’en cas de problème technique, vous devrez trouver de l’aide par vous-mêmes).

Pour l’éditeur de texte, deux options gratuites :

* Sublime Text (<https://www.sublimetext.com/download>)
* Notepad++ (<http://notepad-plus-plus.org/>)

Pour le navigateur :

* Firefox (<https://www.mozilla.org/en-US/firefox/new/>)

Certains exercices plus spécifiques pourraient également nécessiter une connexion à Internet (pour utiliser un validateur en ligne par exemple), mais ce ne sera pas le cas de tous.

Pour info : Firefox et les autres navigateurs web

Dans le cadre de ce cours, nous ne nous intéresserons qu’au navigateur  
Firefox, principalement parce que son utilisation est assez répandue, qu’il se conforme aux normes en vigueur dans le domaine du web et qu’il possède une documentation en ligne facile à consulter.

Pour ce qui concerne JavaScript, la documentation en question se trouve à l’adresse <https://developer.mozilla.org/en-US/docs/Web/JavaScript> !

# (Exo 1) Prise en main de la console Firefox

Objectifs

* Se familiariser avec l’environnement de travail
* Utiliser la console Firefox
* Premières lignes en JavaScript, première définition de fonction
* console.log, prompt et alert

Lancez Firefox puis utilisez le raccourci Ctrl-Shift-K. En bas du browser apparaît la console interactive. Celle-ci permet entre autres d’examiner les éléments HTML de la page web affichée mais également d’exécuter diverses instructions JavaScript à la volée.

## Étape 1 : variables simples

Dans la ligne tout en bas de la console, tapez les commandes suivantes une à une et examinez leur effet. (Assurez-vous que la case « Console » est bien cochée pour rendre visibles les messages envoyés à la console).

**Conseil : prenez le temps de retaper les lignes plutôt que d’utiliser le copier/coller afin de mieux vous familiariser avec le langage.**

let nbJoursSem = 7;

La ligne précédente initialise une variable nbJoursSem à 7. La console répond undefined car vous ne lui avez pas demandé d’effectuer un calcul : vous lui avez juste ordonné de placer la valeur 7 dans une nouvelle variable appelée nbJoursSem.

alert(nbJoursSem);

Une fenêtre s’ouvre, mais ici encore, la console vous répond undefined car l’instruction en question est un appel de fonction (alert) qui ne renvoie aucune valeur.

Voici comment obtenir une autre réponse que undefined : la console vous permet également de connaître la valeur d’une variable. Entrez maintenant la ligne

nbJoursSem

pour afficher la valeur de cette variable. Cela fonctionne également avec une expression, comme par exemple

nbJoursSem + 10

Poursuivez avec la ligne suivante.

console.log("Par semaine : " + (nbJoursSem \* 24) + " heures.");

Puis avec les deux suivantes (utilisez **Shift + Enter** pour passer à la ligne après la première commande sans en demander l’exécution immédiate).

let age = prompt("Quel est votre âge ?");

alert("Âge = " + age);

Notez la différence entre alert et console.log !

## Étape 2 : définition d’une fonction

Entrez la définition de fonction suivante.

function nbJours (nbAns) { return nbAns \* 365; }

La console vous permet de connaître la valeur d’une fonction pour un argument donné en utilisant la syntaxe suivante (entrez les lignes une par une).

nbJours(1);

nbJours(10);

Poursuivez avec les lignes suivantes.

let nb = 4;

alert("Dans " + nb + " ans, il y a " + nbJours(nb) + " jours.");

Entrez ensuite le code suivant (assurez-vous de comprendre de quoi il s’agit).

function entre (nbAns) {  
 let nbJ1 = nbJours(nbAns);  
 let nbJ2 = nbJours(nbAns + 1);  
 return "entre " + nbJ1 + " et " + nbJ2 + " jours";  
}

entre(1);

entre(10);

## Étape 3 : quelques tests de plus

Vous avez vu plus haut que la console vous permet de connaître la valeur d’une variable en entrant tout simplement son nom. Par exemple :

nbJoursSem

pour afficher la valeur de cette variable. Observez le résultat lorsque vous entrez

entre

Testez la fonction « entre » en utilisant l’âge que vous avez entré plus haut.

console.log("Vous avez " + entre(age) + " !");

Observez la dernière ligne affichée par la console : les nombres de jours sont-ils corrects ? Pouvez-vous déterminer d’où vient « l’erreur » ?

**L’ERREUR VIENT DU FAITE QUE 20 + 1 N’EST PAS EGALE A 21 MAIS A 201 !!  
  
Pour résoudre le problème, il faut s’assurer que age est bien un nombre et n’est pas interpréter comme un string.**

**En faisant la conversion Number(age) ou +age**

# (Exo 2) Une Première page HTML-dynamique

Objectifs

* Intégration de scripts JavaScript dans du code HTML
* Production de code HTML via JavaScript
* Introduction à la programmation événementielle

Le point de départ de cet exercice est le document suivant.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8"/>

</head>

<body>

<p><strong>JavaScript</strong> est un langage de scripts compris

par la plupart des navigateurs web. Ce sont ces derniers qui

exécutent le code Javascript.</p>

<p>Grâce à JavaScript, les pages web deviennent</p>

<ul>

<li>interactives,</li>

<li>plus intéressantes,</li>

<li>plus <span id='dyna'>dynamiques</span> !</li>

</ul>

<p>JavaScript est un standard géré par la

<a href=' http://www.ecma-international.org/'>ECMA</a>.</p>

</body>

</html>

## Étape 1 : création du fichier HTML

Créez le fichier HTML, observez son contenu pour vous rappeler la syntaxe du HTML et observez son apparence dans le navigateur.

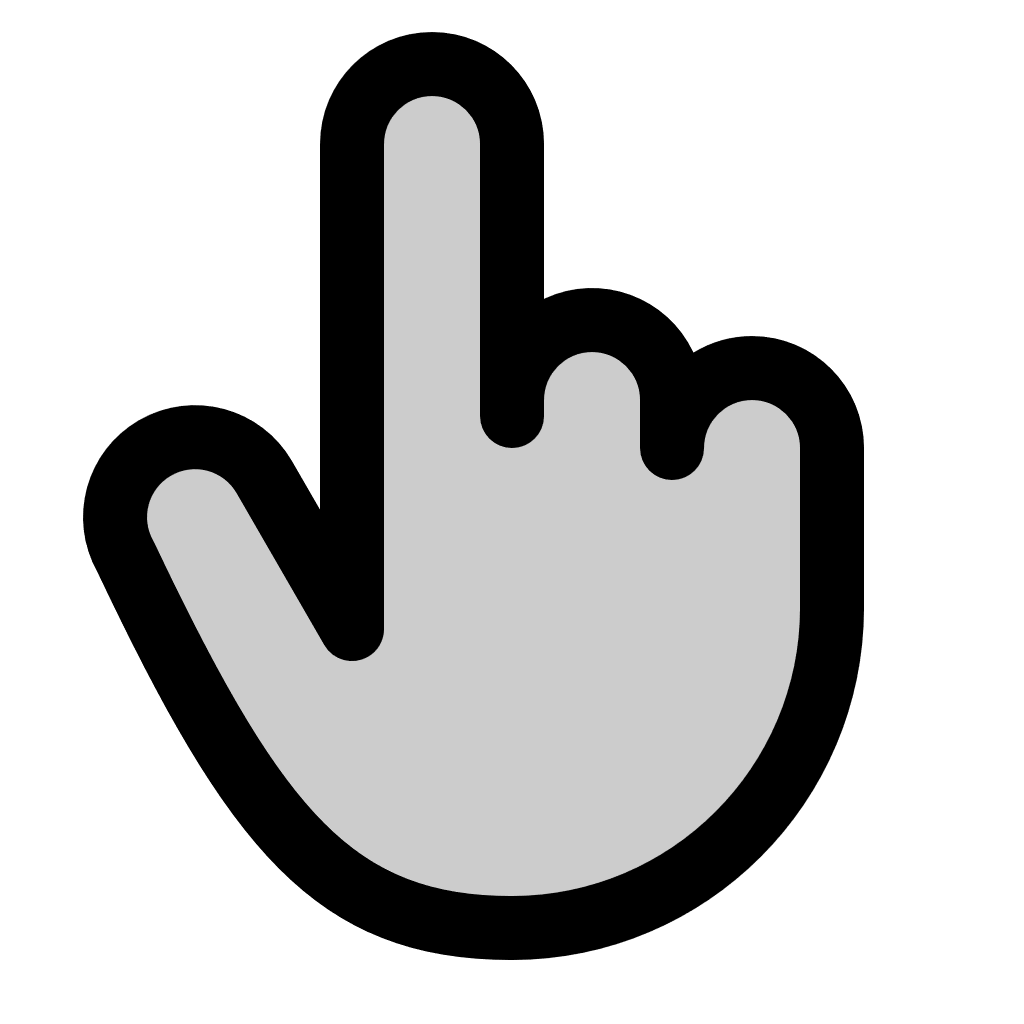
## Étape 2 : script pour ajouter un titre

Éditez le fichier HTML pour ajouter du code Javascript au sein de l’élément <body>, et plus précisément juste avant le dernier paragraphe. Dans ce code,

1. utilisez alert pour afficher le message « Cliquez sur Ok. » ;
2. utilisez document.write pour insérer dans le document HTML le code suivant (qui servira de titre placé juste avant le dernier paragraphe parlant de la ECMA).

<h2>Les origines de JavaScript</h2>

Rechargez la page HTML pour voir les effets du changement (idem dans les étapes suivantes).

**Note importante**

Dans le cadre de ces exercices introductifs, on vous demande d’utiliser la fonction document.write. Celle-ci a un énorme inconvénient : elle permet seulement d’insérer du texte à l’endroit où le script se trouve.

Concrètement, cela signifie que, si on veut insérer du texte via JavaScript à plusieurs emplacements, il faut encombrer le code HTML d’une série de petits scripts qui rendent rapidement le tout illisible. Dans la suite, on verra d’autres méthodes beaucoup plus pratiques, de telle sorte que document.write ne sera quasiment plus utilisé dans les laboratoires suivants ; il s’agit juste d’une méthode aisée pour réaliser les premiers exercices.

## Étape 3 : script pour ajouter l’heure

Notez que, pour ajouter un titre fixe (comme celui de l’étape précédente), il vaut mieux simplement l’intégrer dans le code HTML. L’utilisation de JavaScript est plutôt inutile ici.

Par contre, JavaScript est plus utile si l’objectif est d’ajouter un message qui varie selon certaines circonstances.

Le but de cette étape est d’ajouter un paragraphe en début de document qui indiquera « Bonjour, il est 14h37. » (mais avec l’heure correcte).

Pour ce faire, ajoutez une nouvelle balise <script> tout au début de l’élément <body> et utilisez à nouveau document.write, mais avec un message dépendant de l’heure.

Pour obtenir l’heure, vous pouvez recopier le code suivant (à vous de le compléter pour l’affichage).

let maintenant = new Date ();

let h = maintenant.getHours();

let m = maintenant.getMinutes();

## Étape 4 : fonction changeDyn

Dans l’élément <head> du fichier HTML, ajoutez une nouvelle balise <script> et du code Javascript pour déclarer une fonction nommée changeDyn, avec un argument appelé txt en recopiant le code suivant.

function changeDyn (txt) {

let cible = document.getElementById("dyna");

cible.innerHTML = txt;

}

Rechargez la page HTML… rien n’a changé ? Pourquoi ?

Ouvrez la console (Ctrl + Shift + K) et entrez la ligne suivante et observez son effet.

changeDyn("BELLES");

L’expression document.getElementById("dyna") permet de retrouver l’élément HTML dont l’identificateur est dyna. La syntaxe cible.innerHTML permet de modifier le contenu d’un élément HTML. Ici, le but est de remplacer le contenu du span identifié par dyna par le texte passé en argument.

Notez que le code de la fonction changeDyn aurait pu être écrit en une seule ligne.

function changeDyn (txt) {

document.getElementById("dyna").innerHTML = txt;

}

C’est aussi un bon moment pour revoir les différences entre un identifiant et une classe en HTML/CSS si vous ne vous en souvenez plus !

## Étape 5 : programmation événementielle

Dans le fichier HTML, repérez l’élément identifié par dyna. Ajoutez un attribut à la balise <span>, à savoir :

onclick = "changeDyn('DYNAMIQUES');"

Rechargez la page HTML… et testez les effets de la dernière modification en cliquant sur l’élément HTML correspondant au <span> (c’est-à-dire sur le mot « dynamiques » dans la page HTML).

Pour rappel, cette utilisation de JavaScript est de la programmation événementielle : le code donné dans l’attribut onclick s’exécute lorsqu’un événement se produit. Dans ce cas précis, l’événement en question est un clic sur le <span>.

Notez aussi l’utilisation des apostrophes et des guillemets dans le code donné ci-dessus. En JavaScript, on peut utiliser l’un ou l’autre pour encadrer une chaîne de caractères.

## Étape 6 : fonction toggleDyn

Revenez dans la balise <head> et déclarez-y (avant la définition de fonction) une variable booléenne appelée dynMinuscule et initialisée à true (ce qui signifie que le mot « dynamique » est actuellement écrit en minuscules).

let dynMinuscule = true;

Dans la balise <span>, remplacez le code associé à l’événement onclick par le suivant.

onclick = "toggleDyn();"

Ensuite, dans la balise <head>, définissez une fonction (en fait, une procédure, c’est-à-dire une fonction qui ne renvoie pas de résultat) appelée toggleDyn faisant en sorte qu’à chaque clic sur le mot « dynamiques », celui-ci change entre minuscules et majuscules. Utilisez la variable booléenne pour retenir l’état actuel du mot (minuscule ou majuscule) et savoir s’il faut remplacer le texte par « dynamiques » ou par « DYNAMIQUES ».

Deux notes relatives au « clean code » :

* Faites bien sûr appel à la fonction changeDyn !
* Si vous écrivez du code ressemblant à if (variable == true), considérez-vous comme privé de dessert et de guindailles pendant deux semaines !

# (Exo 3) révision des tables de multiplication

Objectifs

* Révisions HTML et CSS
* Boucles en JavaScript
* Validation d’une entrée
* Scripts dans un fichier externe

Dans cet exercice, vous devrez tout créer à partir de rien : le fichier HTML, éventuellement une feuille de styles CSS et le code JavaScript.

## Étape 1 : création du fichier HTML

Créez un fichier appelé multiplication.html contenant le code suivant.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8"/>

</head>

<body>

<p>Voici le nombre entré :</p>

</body>

</html>

## Étape 2 : demander un nombre

Ajoutez deux scripts JavaScript à ce document HTML :

1. le premier, situé dans la balise <head> et demandant à l’utilisateur d’entrer un nombre (en utilisant prompt) ; le nombre sera stocké dans une variable judicieusement nommée ;
2. le second, après la balise de fin </p>, qui écrira ce nombre dans un nouveau paragraphe <p> (en utilisant document.write).

Par exemple, si le nombre entré est 7, le second script devra produire le code HTML

<p>7</p>

## Étape 3 : le faire avec style

Afin de mettre en évidence le nombre choisi (qui servira de base pour une table de multiplication par la suite), on peut lui ajouter un style (c’est aussi l’occasion de revoir un peu de CSS). Créez un fichier multiplication.css contenant le code suivant.

.nombre {

text-align: center;

background-color: blue;

color: white;

font-weight: bold;

}

Ajoutez également la ligne suivante dans la partie head du document HTML afin de lier la feuille de style.

<link rel="stylesheet" type="text/css" href="multiplication.css"/>

Modifiez le script JavaScript pour que le nombre entré soit affiché en utilisant ce style. Le code HTML produit par Javascript devra être similaire à

<p class="nombre">7</p>

**Note.** Comme indiqué plus haut, en JavaScript, on peut utiliser indifféremment des guillemets ou des apostrophes pour encadrer les chaînes de caractères. C’est tout particulièrement intéressant quand on veut écrire dans le document du code HTML contenant des guillemets. Une autre option consiste à utiliser un caractère d’échappement (le backslash) en notant \" à l’intérieur d’une chaîne encadrée par des guillemets (ou \' à l’intérieur d’une chaîne encadrée par des apostrophes).

## Étape 4 : boucle Javascript (version liste)

C’est parti pour la table de multiplication…

Complétez le script d’affichage (celui qui se trouve dans la partie body) pour qu’après avoir sorti le nombre choisi, il affiche la table de multiplication de celui-ci. Dans un premier temps, vous pouvez sortir la table de multiplication sous la forme d’une liste HTML.

Pour rappel, une liste HTML se compose avec les balises <ul> et </ul>. À l’intérieur de ces balises, chacun des éléments est encadré par les balises <li> et </li> (li = list item). Par exemple :

<ul>

<li>Élément 1</li>

<li>Élément 2</li>

<li>Élément 3</li>

</ul>

A screenshot of a math test

Description automatically generatedDans la situation qui nous préoccupe, le but est d’obtenir la sortie ci-contre (si le nombre entré est 7).

Pour ce faire, vous devrez étoffer votre script JavaScript en utilisant une boucle. Les boucles JavaScript utilisent la même syntaxe que celles de C ou de Java ; on peut même déclarer la variable utilisée dans la boucle à l’intérieur de celle-ci. Voici un exemple dont vous pouvez vous inspirer.

for (let age = 18 ; age < 25 ; age++) {

alert("age = " + age);

}

Pour voir l’effet exact de ce bout de code donné en exemple, vous pouvez tout simplement le copier/coller dans la console interactive de Firefox !

## Étape 5 : boucle Javascript (version tableau)

A table with numbers and a yellow background

Description automatically generated with medium confidenceDans un second temps, sortez la table de multiplication sous la forme d’un tableau HTML.

Définissez, dans la feuille de styles, des règles de style correspondant aux sélecteurs suivants :

* th : pour les titres des colonnes (par exemple : fond grisâtre, écriture bleue, gras)
* tr : pour les lignes (par exemple : fond blanc)
* table tr:nth-child(odd) : pour les lignes impaires (par exemple : fond jaune)
* tr td:nth-child(3) : pour les résultats (les 3e valeurs) de chaque ligne (par exemple : gras)

Assurez-vous que vous comprenez bien la signification de chacun de ces quatre sélecteurs !

## Étape 6 : un code en ordre

Mettez un peu d’ordre dans votre code.

Généralement, on place toutes les définitions de fonctions Javascript dans la balise <head> du document HTML. À l’intérieur de la balise <body>, on ne trouve donc plus que des appels de fonctions.

Transformez le code d’affichage du nombre choisi et le code d’affichage de la table de multiplication en deux fonctions qui seront définies (mais pas appelées) dans <head>. Placez les appels adéquats au bon endroit dans <body>.

## Étape 7 : valeurs inattendues

Testez votre script Javascript lorsqu’on lui donne des données inattendues, comme par exemple :

* rien (on appuie directement sur Enter sans entrer de nombre) ;
* un mot (« pomme ») au lieu d’un nombre ;
* « 12+4 »
* « 5doigts »

## Étape 8 : validation du nombre entré

La commande prompt renvoie la réponse donnée par l’utilisateur sous la forme d’une chaîne de caractères. Rien ne dit qu’il s’agit d’un nombre valable !

Dans de nombreuses applications sur le web, il est nécessaire de valider les informations entrées par l’utilisateur. Ce mot peut recouvrir toute une série de vérifications plus ou moins complexes : validité d’une adresse mail qui doit être écrite au bon format, validité d’un mot de passe qui doit respecter certaines conditions de sécurité…

Dans un premier temps, convertissez la réponse de l’utilisateur en nombre. Pour ce faire, vous pouvez utiliser la fonction Number : Number(x) est la conversion de « x » (quel que soit son type) en un nombre.

Javascript prévoit trois « nombres » particuliers :

* NaN, qui signifie « not a number » : c’est la réponse donnée quand on tente de transformer une chaîne comme "bonjour" en un nombre ;
* Infinity, qui est le résultat de 5/0 par exemple ; et
* -Infinity, qui est le résultat de -3/0.

Pour observer ces valeurs, vous pouvez par exemple rentrer les trois lignes suivantes dans la console interactive de Firefox (une par une).

"pomme" \* 4

71 / 0

-12 / 0

La fonction booléenne isFinite(x) permet de vérifier que x est un nombre « fini », c’est-à-dire une valeur numérique autre que NaN, Infinity et –Infinity. Modifiez le script de <head> pour qu’il demande à l’utilisateur de rentrer un nombre jusqu’à ce que ce dernier le fasse (si l’utilisateur encode autre chose, on lui pose la question à nouveau).

Dans votre script, vous pourrez utiliser une boucle while (la syntaxe est identique à celle qu’on emploie en C ou en Java).

## Étape 9 : Javascript en mode externe

Pour terminer, passez à une version externe (fichier séparé) du code Javascript.

Extrayez tout le code Javascript de <head> sous la forme d’un fichier séparé (multiplication.js) puis remplacez-le par le code HTML permettant d’intégrer ce fichier.

# (Exo 4) compteur, boutons et liens

Objectifs

* Programmation événementielle
* Variable globale à un script

Cet exercice est moins détaillé que le précédent : ici, c’est à vous à voir où placer les scripts et comment les organiser.

## Étape 1

Créez un fichier html comportant le code suivant. Observez son affichage.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8"/>

</head>

<body>

<p>Combien de fois aurez-vous le courage de cliquer sur le bouton ?</p>

<button>Click me!</button>

<p>Cliquez <a>ici</a> pour remettre le compteur à zéro !</p>

</body>

</html>

## Étape 2

Le document comporte deux éléments potentiellement réactifs mais ils ne font rien pour l’instant. Il s’agit du bouton « Click me ! » et du lien sur le mot « ici » (qui, vu qu’il ne mène nulle part, n’est sans doute même pas représenté comme un lien par Firefox).

Dans un premier temps, pour tester l’association de scripts JavaScript à ces éléments, modifiez le fichier HTML pour que

* cliquer sur le bouton ouvre une fenêtre indiquant « Vous avez cliqué sur le bouton ! » (vous pouvez utiliser la fonction alert) ;
* cliquer sur le lien fasse apparaître une fenêtre indiquant « Vous avez cliqué sur le lien ! ».

Comme ces deux opérations sont assez courtes, vous pouvez incorporer le code directement dans les balises html <button> et <a> (dans un attribut onclick pour <button> et dans un attribut href pour <a>).

## Étape 3

Pour faciliter les éditions qui vont suivre, mettez le code de ces actions dans des fonctions séparées (appelées par exemple clicBouton() et clicLien()) et, dans les balises, appelez simplement la fonction adéquate.

## Étape 4

Si vous avez observé le contenu du fichier HTML, vous aurez sans doute compris que le but final de cet exercice est de réaliser un compteur de clics sur le bouton (appelé par exemple nbClicks — c’est beaucoup plus parlant que juste compteur !).

Chaque fois que vous cliquerez sur le bouton, il devra afficher (via alert) le nombre de clics déjà effectués.

D’autre part, en cliquant sur le lien, ce nombre devra être remis à zéro.

Modifiez les fonctions clicBouton() et clicLien() pour que tout se passe de la sorte ! N’oubliez pas de déclarer la variable nbClicks en-dehors des fonctions !

# (Exo 6) Mini-tableur

Objectifs

* Exercice récapitulatif

Construisez un document HTML (avec un ou plusieurs scripts JavaScript) qui permettra à l’utilisateur de rentrer des nombres (les uns après les autres, en entrant une chaîne vide ou autre chose qu’un nombre pour terminer). Le document devra produire un tableau à 5 colonnes comportant une ligne de titres (voir ci-dessous) et une ligne de plus pour chaque nombre entré.

Chaque fois que l’utilisateur aura entré un nombre, le script écrira une nouvelle ligne (à l’aide de document.write) reprenant (a) le numéro du nombre entré, (b) le nombre entré lui-même, (c) le minimum des nombres entrés jusque-là, (d) le maximum des nombres entrés jusque-là et (d) la somme des nombres entrés jusque-là.

Par exemple, si l’utilisateur entre les nombres -3, 17, -5 et 20 (avant d’appuyer simplement sur Enter), le document devrait afficher le tableau suivant.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Nombre** | **Min** | **Max** | **Somme** |
| 1 | **-3** | **-3** | **-3** | **-3** |
| 2 | 17 | **-3** | 17 | 14 |
| 3 | **-5** | **-5** | 17 | 9 |
| 4 | 20 | **-5** | 20 | 29 |

Le script principal sera donc une boucle qui demandera à l’utilisateur une entrée puis qui, en fonction de cette entrée, ajoutera une ligne au tableau (via document.write) ou se terminera (n’oubliez pas de fermer les balises HTML laissées ouvertes).

Dans le tableau, les nombres positifs (ou nuls) seront affichés en bleu et les nombres négatifs en gras rouge. Pour réaliser cet affichage de manière « organisée », il sera sans doute préférable de vous baser sur les deux styles CSS suivants (à placer dans une balise <style> située dans l’en-tête du document).

.negatif { color : red ; font-weight : bold }

.positif { color : blue }

Pour vous faciliter la tâche, créez une fonction afficheValeur() prenant comme argument le nombre à afficher et renvoyant le code HTML pour une cellule (<td>) de tableau affichant ce nombre (y compris le style à utiliser : classe negatif ou classe positif).

N’hésitez pas à comparer votre code final avec celui d’autres étudiant(e)s et/ou à le faire commenter par un enseignant !