

Module LPB

...

I. Le javascript

Début de la séance à 18h!!!

1. Introduction générale au Js

Dans le cadre de cette formation, on étudiera le Javascript récent, supporté par les dernières versions en date des principaux navigateurs: Chrome, Firefox, Safari et Edge (exit IE)
Les exemples et les exercices se baseront sur la version 5 d'html.

On a délibérément choisi de ne pas résumer cette formation à une longue description de toutes les fonctionnalités offertes par le langage (liste de fonctions, d'objets, de propriétés, de méthodes...) comme c'est souvent le cas, assortie d'exemples que la plupart des gens se contentent malheureusement de recopier sans les comprendre. La description de ces fonctionnalités est largement disponible sur de nombreux sites de référence

On a préféré décrire en détail le langage lui même, avec toutes ses spécificités et particularités qui le distingue des autres langages. Une meilleure compréhension du langage est en effet la clé pour arriver à se débrouiller pour écrire ses propres programmes, ou à copier et adapter de manière intelligente ceux disponibles sur le web

Où trouver de l'aide?

- <http://www.w3schools.com/js/default.asp> (facile mais peu précise)
- <https://developer.mozilla.org/fr/docs/Web/JavaScript> (très précise mais compliquée)
- <http://www.ecma-international.org/ecma-262/10.0/> (incompréhensible mais très rigoureuse)

Javascript: c'est quoi ?

Le javascript est un **langage de programmation** (langage de scripting) **orienté objet**

- Il a été créé pour enrichir des pages web et leur donner un comportement dynamique (qui va interagir avec l'utilisateur)

- Il est de plus en plus souvent utilisé en dehors des pages web:

- Javascript coté serveur (CommonJS, Node.JS...)
- Applications pour smartphone (PhoneGap...)
- Macros dans OpenOffice
- Actionscript dans les animations Flash
- Widgets...

- La syntaxe de base est similaire à celle des langages C, C++ ou Java (en plus simple)

C'était à l'origine un langage propriétaire développé par Netscape (conjointement avec Sun). Il a rapidement été cédé à l'ECMA (European Computer Manufacturers Association) pour qu'il en fasse une **norme** afin qu'il soit compatible avec tous les navigateurs

La version normalisée s'appelle l'**ECMAScript** (version 11, actuellement-> ES2020)

Javascript est un langage interprété

Le Javascript se classe dans la catégorie des langages de programmation **interprétés**

•**langages compilés:** le code source est analysé par un programme appelé **compilateur** qui va générer du code binaire que l'ordinateur sera capable d'exécuter (attention! ce code binaire sera différent d'un ordinateur à l'autre et nécessite d'être recompilé pour chaque ordinateur). Les langages comme le C ou le C++ sont des langages compilés

•**langages précompilés:** le code source est compilé partiellement, dans un code plus simple mais qui n'est pas du code binaire (ce code est valable pour tous les types de machines). Ce code intermédiaire sera ensuite interprété et exécuté par une **machine virtuelle** (propre à chaque ordinateur). Les langages comme le C# ou le Java sont des langages précompilés

•**langages interprétés:** il n'y a pas de compilation ni de précompilation au préalable. Le code source reste tel quel. Si on veut l'exécuter, il faut faire appel à un **interpréteur** qui se chargera de l'analyser et de réaliser les actions qu'il contient. Le langage Javascript est un langage interprété

Chaque navigateur possède son interpréteur Javascript (Chakra chez Microsoft, SpiderMonkey - et ses dérivés TraceMonkey, JägerMonkey... - chez Mozilla, JavaScriptCore - et ses dérivés SquirrelFish, Nitro... - chez Apple, V8 chez Chrome...)

Remarque: pour améliorer les performances, la plupart de ces interpréteurs procèdent à une pseudo précompilation "à la volée" pendant l'exécution (si le même code doit être réexécuté, il sera beaucoup plus rapide)

Javascript est un langage orienté Objet

Le Javascript se classe dans la catégorie des langages de programmation **orientés objets**

Cela veut dire que le langage va manipuler des données sous la forme d'**objets**. Chaque objet possède des caractéristiques particulières classées entre **propriétés** et **méthodes**

Le langage fournit des objets de base comme des chaînes de caractères, des images, des dates...

Objet de type String

.length
.toLowerCase()
.toUpperCase()
...

Objet de type Image

.src
.width
.height
...

Objet de type Date

.getDate()
.getMonth()
.getFullYear()
...

Il est également possible de créer ses propres objets (ce qui simplifie le code si ces objets sont bien choisis)

Objet de type Voiture

.marque
.modèle
.année
.conducteur
.changerDeConducteur()
...

Objet de type Conducteur

.nom
.prénom
.voiture
.changerDeVoiture()
...

La balise script

Le code Javascript peut se placer entre deux balises <script> et </script>

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>exemple javascript</title>
7  </head>
8  <body>
9      <script>
10         const nom = "Vilain";
11         const prenom = "Guy";
12
13         alert("Bonjour" + nom + " " + prenom + " !");
14     </script>
15
16  </body>
17  </html>
```

La balise script

Plusieurs balises peuvent apparaître à différents endroits dans la page html
Elles seront traitées au fur et à mesure que le html est traité et affiché par le navigateur

```
> index.html > html > body
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>exemple javascript</title>
7       <script>
8         const nom = "Duck"
9         const prenom = "Donald"
10      </script>
11    </head>
12    <body>
13      <script>
14        alert("Bonjour " + prenom + " " + nom + " !");
15      </script>
16    </body>
17  </html>
```

Il est important de comprendre que ces deux parties de codes seront exécutées en même temps que l'affichage de la page, l'une après l'autre.

La balise script

Il faut s'assurer que les ressources existent avant de les utiliser

Il faut veiller à ne pas utiliser une ressource qui n'existe pas encore (comme une variable, par exemple)
Cet exemple générera une erreur (les deux variables n'existent pas au moment où on essaye de les utiliser)

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>exemple javascript</title>
7   <script>
8       alert("Bonjour "+prenom+" "+nom+" !");
9   </script>
10  </head>
11  <body>
12      <script>
13          var nom="Mouse";
14          var prenom="Mickey";
15      </script>
16  </body>
17 </html>
```

La balise script

Dans les anciennes versions de Javascript, il était recommandé de masquer le code Javascript à l'aide de commentaires (remarquez le // devant le -->):

```
<script>  
<!--  
... code Javascript ...  
// -->  
</script>
```

L'attribut **type** était obligatoire dans les anciennes versions d'html. Il ne l'est plus en html 5 (il vaut par défaut `text/javascript`):

```
<script type="text/javascript">  
... code Javascript ...  
</script>
```

L'attribut **language** assurait la compatibilité dans les vieux navigateurs. Il n'est plus supporté actuellement:

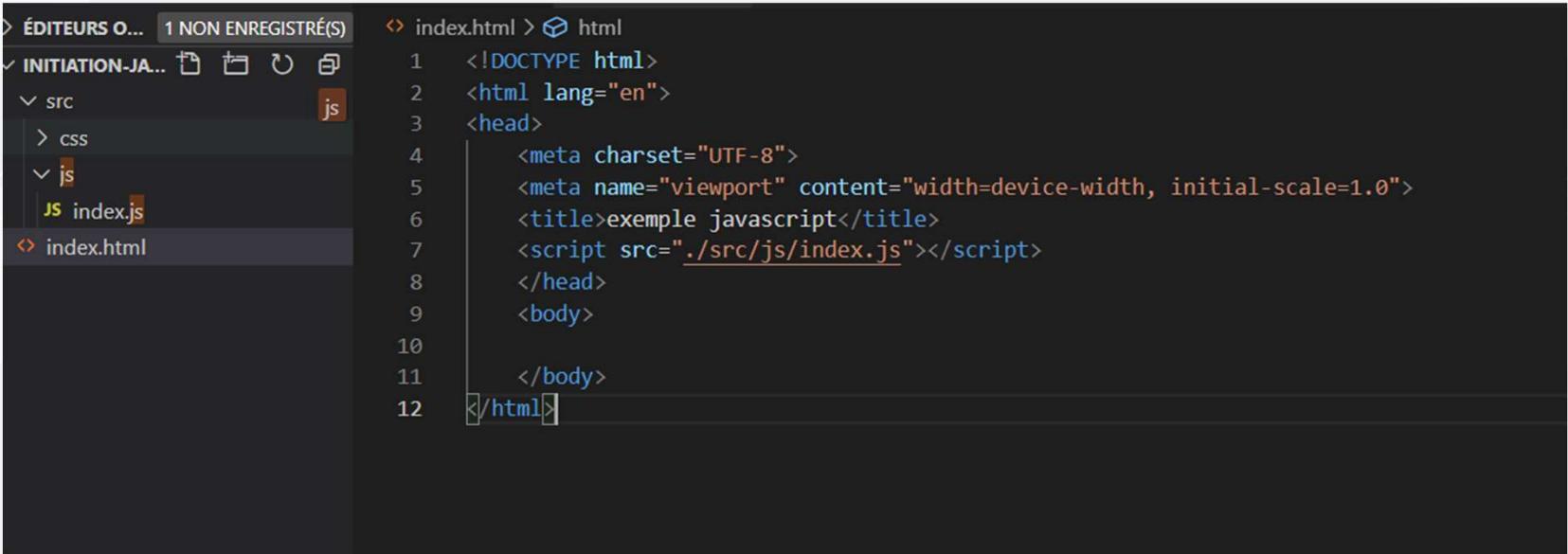
```
<script language="JavaScript1.2">  
... code Javascript ...  
</script>
```

Placer le code Js dans un fichier séparé

Il est possible, et souvent conseillé, de placer les instructions dans un document séparé dont l'extension est ".js"
La balise <script> doit être vide, et l'url du document javascript doit être mentionnée dans l'attribut **src** (url
absolue ou url relative)

Ce document ".js" ne peut contenir que du Javascript (pas de balises html, ni de balises <script>)

L'avantage est que le code peut être partagé par plusieurs pages html et peut bénéficier de la mise en cache du
document Javascript



The screenshot shows a code editor interface with a dark theme. On the left is a sidebar titled "ÉDITEURS O..." showing "1 NON ENREGISTRÉ(S)". Below it is a tree view of a project structure named "INITIATION-JA...". The structure includes a "src" folder containing "css" and "js" subfolders. Inside "js" is a file named "index.js". The main editor area displays the content of "index.html". The code is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>exemple javascript</title>
    <script src=".src/js/index.js"></script>
</head>
<body>
</body>
</html>
```

Où placer les balises script ?

Certaines personnes conseillent de placer les balises <script> dans le <head> de la page, d'autres à la fin du <body>... difficile de s'y retrouver.

Chargé dans le <head>, le code est accessible dans le reste de la page, mais cela peut ralentir l'affichage si la librairie est volumineuse.

Chargé dans le <body>, le code ne ralentit pas l'affichage de la page, mais on ne pourra l'utiliser qu'une fois la page chargée.

Le meilleur conseil:

- suivre les consignes données par les librairies quand on en utilise une.
- savoir ce qu'on fait pour le reste (identifier le meilleur endroit ou charger le code)

Où placer les balises script ?

Certaines personnes conseillent de placer les balises <script> dans le <head> de la page, d'autres à la fin du <body>... difficile de s'y retrouver.

Chargé dans le <head>, le code est accessible dans le reste de la page, mais cela peut ralentir l'affichage si la librairie est volumineuse.

Chargé dans le <body>, le code ne ralentit pas l'affichage de la page, mais on ne pourra l'utiliser qu'une fois la page chargée.

Le meilleur conseil:

- suivre les consignes données par les librairies quand on en utilise une.
- savoir ce qu'on fait pour le reste (identifier le meilleur endroit ou charger le code)

Quelques attributs de la balise script ?

L'attribut `async` (`async`, `async=""` ou `async="async"`) permet d'exécuter le code Javascript externe en mode asynchrone:

```
<script src="moncode.js" async></script>
```

Si l'attribut `async` n'est pas présent, l'attribut `defer` (`defer`, `defer=""` ou `defer="defer"`) permet de différer l'exécution du code Javascript après le chargement de la page:

```
<script src="moncode.js" defer></script>
```

L'attribut `charset` permet de définir le jeu de caractères utilisé dans le document Javascript externe:

```
<script src="moncode.js" charset="UTF-8"></script>
```

La balise No script

La balise `<noscript>` permet d'afficher un message en html si le navigateur ne supporte pas le Javascript ou si celui-ci est désactivé

```
10
11      <noscript>
12          |      <p class="important">Vous devez activer le Javascript pour visualiser correctement cette page</p>
13      </noscript>
14
```

Placer du code javascript dans un gestionnaire d'événement

De nombreuses balises html peuvent être associées à des **gestionnaires d'événement**. Ce sont des attributs qui contiennent du code Javascript qui sera exécuté lorsqu'un événement particulier se produira (click ou déplacement de souris, frappe au clavier, valeur entrée dans un formulaire, etc)

Ces attributs s'appellent **onload**, **onunload**, **onmouseover**, **onmouseout**, **onclick**, **onkeypress**, **onchange**...

Dans cet exemple, le gestionnaire d'événement **onclick** est exécuté à chaque fois que l'utilisateur clique sur le lien. Celui-ci fait appel à la fonction `confirm(...)` qui affiche une boîte de dialogue et qui retourne true ou false en fonction du bouton choisi par l'utilisateur (ok ou cancel).

L'action par défaut, qui est de suivre le lien hypertexte mentionné, sera désactivée si la valeur false est renvoyée.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>exemple javascript</title>
7      <script src=".src/js/index.js"></script>
8  </head>
9  <body>
10     <p>Visitez le site du
11        <a href="http://www.lesoir.be/" onclick="return confirm('êtes-vous sûr ?')">journal Le Soir</a>.
12    </p>
13  </body>
14 </html>
```

L'environnement d'exécution et l'objet Window

Lors de l'affichage d'une page, le navigateur va créer un environnement d'exécution pour le code Javascript, qui sera disponible durant tout le temps que la page restera affichée.

Cet environnement est matérialisé par un objet appelé **window**.

Dès qu'une page est affichée, cet objet est créé et restera le même jusqu'au moment où une nouvelle page viendra remplacer l'ancienne. Les variables et les fonctions globales sont stockées dans cet objet window. Une variable globale ou une fonction globale n'est rien d'autre qu'une propriété de cet objet.

Dans cet exemple, les variables seront encore disponibles lorsque l'utilisateur cliquera sur le lien.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>exemple javascript</title>
7  <script>
8      var nom="Mouse";
9      var prenom="Mickey";
10     </script>
11 </head>
12 <body>
13     <p>Dites <a href="#" onclick="alert('Hello '+prenom+' '+nom+' !'); return false;">hello</a>.</p>
14 </body>
15 </html>
```

On aurait pu également écrire
alert('Hello
' + window.prenom +
' ' + window.nom + ' !'),
pour bien montrer que ces deux variables globales sont des propriétés de l'objet **window**.

Création de gestionnaire d'événement en Js

Les gestionnaires d'événement peuvent également être créés en Javascript

Dans cet exemple, on crée un gestionnaire d'événement onclick sur toutes les balises <a> trouvées dans la page web.

« **document** » est une variable (en réalité, une propriété de l'objet **window**), qui représente le document HTML chargé dans le navigateur. C'est le deuxième objet le plus important, après **window**.

Cet objet est défini dans une norme appelée **DOM** (Document Object Model).

La méthode « **getElementsByName** » renvoie la liste de toutes les balises html, elles-mêmes représentées par des objets définis dans DOM, dont le nom est égal à celui donné en paramètre (ce nom est automatiquement transformé en lettres majuscules).

(Voir exemple01.html)

```
8   <script>
9     function defineLinks()
10    {
11      var links = document.getElementsByTagName("a");
12      for (var i=0; i<links.length; i++)
13      {
14        links.item(i).onclick =  function() { return confirm('êtes-vous sûr ?'); };
15      }
16    }
17  </script>
18 </head>
19 <body onload="defineLinks()">
20   <p>Visitez le site du <a href="http://www.lesoir.be/">journal Le Soir</a>.</p>
21 </body>
```

Exécution de code Js dans un lien hypertexte

Il est possible d'exécuter du code Javascript à l'aide d'un lien hypertexte dont le protocole est **javascript**:

```
<body>
|   <a href="javascript:alert('hello world')">test</a>.
|</body>
```

Si le code Javascript retourne une valeur, elle sera traitée comme une chaîne de caractères contenant du code html à afficher en lieu et place de la page courante

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  |   <title>exemple javascript</title>
7  <script>
8  |   prenom='Mickey';
9  |   nom='Mouse';
10 |</script>
11 </head>
12 <body>
13 |   <a href="javascript:'<p>Hello '+nom+' '+prenom+' !</p>'">hello</a>.
14 </body>
15 </html>
```

Pratique 01: afficher un texte dans une langue choisie par le user

Créez une page html qui présente trois liens hypertextes (balise <a>): hello, français et anglais.

En cliquant sur le lien hello, le message "Bonjour [votre prénom] [votre nom] !" doit apparaître dans une boîte de dialogue, grâce à la fonction alert(...). Si l'anglais est sélectionné, c'est le message "Hello [votre prénom] [votre nom] !" qui doit apparaître.

Les deux liens français et anglais doivent servir à sélectionner la langue.

Utilisez des variables pour stocker la langue, votre nom et votre prénom.

2. Les bases de Javascript

Les objets principaux et les types d'objets à connaître

Javascript regorge d'objets créés automatiquement lors du chargement de la page web. Les principaux à connaître et que nous utiliseront probablement durant la formation sont:

Les types d'objets natifs

Le langage Javascript utilise une quantité de types d'objet natifs (prédéfinis dans le langage)

String	pour les chaînes de caractères
Number	pour les valeurs numériques
Boolean	pour les valeurs booléennes
Array	pour les tableaux
Object	pour les objets
Function	pour les fonctions
Event	pour les événements
...	et bien d'autres

2. Les bases de Javascript

Les objets principaux et les types d'objets à connaître

Les objets du BOM (Browser Object Model)

Le langage Javascript intégré dans un navigateur utilise une quantité d'objets prédéfinis utilisables en Javascript pour contrôler le navigateur et la page web affichée

window	représente la fenêtre du navigateur. Il est le père de tout ! (les variables globales sont des propriétés de cet objet, les fonctions globales sont des méthodes de cet objet...)
document	représente la page chargée dans le navigateur, c'est un objet de type HtmlDocument qui dérive du type Document de DOM
JSON	pour parser des données au format JSON et créer les objets Javascript correspondants
...	et bien d'autres

Les types d'objets de DOM (Document Object Model)

DOM est une interface normalisé pour décrire tous les types d'objets utilisés pour modéliser une page html ou un document xml

Document	pour représenter un document xml ou html. L'objet document est du type HtmlDocument qui dérive de Document
Element	pour représenter un élément (ou balise) xml ou html. Toutes les balises html, par exemple, sont représentées par un objet du type HTMLElement qui est lui-même du type Element
Attr	pour représenter un attribut xml ou html. Tous les attributs des balises html, par exemple, sont représentés par un objet du type Attr
Text	pour représenter un noeud texte xml ou html. Toutes les chaînes de caractères contenues dans des balises html, par exemple, sont représentées par un objet du type Text
...	et bien d'autres

2. Les bases de Javascript

Les objets principaux et les types d'objets à connaître

Les types d'objets pour l'html

Toutes les balises html présentes dans une page vont donner lieu à la création d'un objet en Javascript. Il s'agit d'un objet de type **Element** (DOM) augmenté de propriétés et de méthodes supplémentaires

HtmlDocument	pour représenter un document, dérive du type Document
HtmlElement	pour représenter un élément, dérive du type Element
HtmlImageElement	pour représenter une image, dérive du type HtmlElement
HtmlFormElement	pour représenter un formulaire, dérive du type HtmlElement
HtmlInputElement	pour représenter un <input> dans un formulaire, dérive du type HtmlElement
HtmlSelectElement	pour représenter une liste déroulante, dérive du type HtmlElement
HtmlOptionElement	pour représenter une option dans une liste déroulante, dérive du type HtmlElement
Style	pour représenter les styles CSS associés à une balise html
...	et bien d'autres

Les types d'objets supplémentaires

De nombreux types d'objet supplémentaires existent (et les nouvelles versions de Javascript ne cessent d'en ajouter). Parmi-eux:

2. Les bases de Javascript

Les objets principaux et les types d'objets à connaître

Les types d'objets supplémentaires

De nombreux types d'objet supplémentaires existent (et les nouvelles versions de Javascript ne cessent d'en ajouter). Parmi-eux:

Date	pour les dates&heures
Math	pour les manipulations mathématiques
RegExp	pour les expressions régulières
XMLHttpRequest	pour créer et gérer des connexions en AJAX vers un serveur distant
...	et bien d'autres

L'objet Window

L'objet **window** est l'objet principal de Javascript

C'est un objet qui représente à la fois le navigateur, le document html chargé dans le navigateur, le javascript lui-même...

Il est à la base de pratiquement toutes les informations que l'on va traiter en Javascript

Il représente l'environnement global de tout programme Javascript. Toutes les variables et toutes les fonctions créées dans cet environnement global seront en réalité des propriétés et des méthodes de cet objet **window**

<code>.document</code>	représente le document html chargé dans le navigateur (du type <code>HtmlDocument</code> qui dérive de <code>Document</code>)
<code>.screen</code>	renseigne sur l'écran de l'utilisateur (taille...)
<code>.location</code>	renseigne sur l'adresse de la page chargée dans le navigateur et permet de charger une autre page
<code>.history</code>	manipule l'historique des pages qui ont été chargées dans le navigateur
<code>.navigator</code>	renseigne sur le type de navigateur
<code>.localStorage</code> <code>.sessionStorage</code>	permet de sauver/récupérer des objets Javascript localement pour toutes les pages partageant le même domaine
<code>.screenX</code> <code>.screenY</code>	coordonnées de la fenêtre du navigateur relatives à l'écran
<code>.innerWidth</code> <code>.innerHeight</code>	largeur et hauteur internes en pixels de la fenêtre du navigateur (largeur et hauteur réellement utilisables)
<code>.outerWidth</code> <code>.outerHeight</code>	largeur et hauteur externes en pixels de la fenêtre du navigateur (incluant les scrollbars et les toolbars)
<code>.pageXOffset</code> <code>.pageYOffset</code>	déplacement de la page par rapport à la fenêtre (scrolling) le long de l'axe horizontal et de l'axe vertical
<code>.closed</code>	indique si la fenêtre a été fermée (la fenêtre n'existe plus, mais l'objet <code>window</code> qui la représentait existe toujours)
<code>.frames</code> <code>.length</code>	un tableau reprenant les <code><frame></code> éventuelles, et le nombre de ces frames
<code>.top</code>	l'objet <code>window</code> de la fenêtre de plus haut niveau dans le cas, par exemple, où le document est découpés en frames (chaque frame aura son propre objet <code>window</code>)
<code>.opener</code>	l'objet <code>window</code> de la fenêtre qui a ouvert la fenêtre courante dans le cas, par exemple, où une fenêtre a été ouverte par un <code>window.open(...)</code>
<code>.parent</code>	l'objet <code>window</code> de la fenêtre parent (ou la fenêtre elle-même si elle n'a pas de parent) dans le cas, par exemple, où le document est découpés en frames (chaque frame aura son propre objet <code>window</code>)
<code>.self</code>	l'objet <code>window</code> de la fenêtre courante
<code>.window</code>	l'objet <code>window</code> lui-même
...	

Exemple utilisation de la propriété Window.location

La propriété .location de l'objet window renseigne sur l'adresse - l'URL - de la page chargée dans le navigateur. On peut obtenir l'adresse complète, ou l'une de ses composantes individuelles: le protocole (http:), l'adresse du serveur (http://127.0.0.1), le numéro de port (:5500), etc.

Il s'agit d'un objet de type **Location** dont on trouvera toutes les caractéristiques dans la documentation du langage. Cet objet possède entre-autres une propriété .href qui donne l'URL complète de la page
Cette propriété est accessible en lecture (on peut lire l'adresse), mais également en écriture (on peut y écrire une nouvelle adresse)

La fait de modifier cette adresse va automatiquement charger une nouvelle page dans le navigateur. La page courante sera automatiquement remplacée par la page se trouvant à la nouvelle adresse
Ce comportement est très fréquent en Javascript. Une modification d'un objet en Javascript déclenche très souvent un changement dans le navigateur ou dans la page html affichée par le navigateur

Voir Exemple02.html

Exemple utilisation de la propriété Window.history

La propriété .history de l'objet window renseigne sur l'historique des pages qui ont été visualisées dans le navigateur

Comme bien souvent, tout ne sera pas possible pour des questions de sécurité et de confidentialité. Vous pouvez, par exemple, revenir en arrière dans l'historique, mais vous ne pouvez pas connaître l'adresse des pages dans cet historique

Cette propriété est un objet du type **History**

La méthode la plus intéressante s'appelle history.back() qui permet de revenir à la page précédente

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>exemple javascript</title>
7  </head>
8  <body>
9      <p><a href="javascript:window.history.back()">revenir en arrière</a></p>
10 </body>
11 </html>
```

Les méthodes principales de l'objet window

< ⌂ >

.alert(...)	affiche une boîte de dialogue pop-up pour afficher un message à l'écran
.confirm(...)	affiche une boîte de dialogue pop-up afin de demander une confirmation à l'utilisateur
.prompt(...)	affiche une boîte de dialogue pop-up afin de demander à l'utilisateur d'entrer une valeur
.open(...)	ouvre une nouvelle fenêtre ou un nouvel onglet
.close()	ferme la fenêtre courante
.focus()	donne le focus à la fenêtre courante
.blur()	enlève le focus à la fenêtre courante
.print()	imprime le contenu de la fenêtre
.setTimeout(...)	lance l'exécution de code Javascript de manière différée dans le temps
.clearTimeout(...)	supprime une exécution différée demandée par .setTimeout(...)
.setInterval(...)	lance l'exécution de code Javascript à intervalles réguliers
.clearInterval(...)	supprime une exécution à intervalles réguliers demandée par .setInterval(...)
.moveTo(...)	déplace la fenêtre courante à une position donnée
.moveBy(...)	déplace la fenêtre courante d'une distance donnée
.resizeTo(...)	redimensionne la fenêtre courante à une taille donnée
.resizeBy(...)	redimensionne la fenêtre courante d'une distance donnée
.scrollTo(...)	scrolle la fenêtre courante à une position donnée
.scrollBy(...)	scrolle la fenêtre courante d'une distance donnée
.postMessage(...)	envoie un message à un autre objet window
...	

Exemple: la méthode Window.alert()

Cette méthode permet d'afficher un message dans une boîte de dialogue (pop-up). Le message va s'afficher avec un bouton OK permettant à l'utilisateur de fermer ce pop-up
alert(message)

Son utilisation est un peu tombée en désuétude, mais elle reste très utile pour mettre au point un programme Javascript (pour faire apparaître la valeur de certaines variables ou s'assurer que votre programme passe bien par certains points, par exemple)

On l'utilisera dans de nombreux exemples qui vont suivre pour sa facilité de mise en œuvre. Dans une application réelle, on utilisera bien évidemment d'autres techniques plus modernes pour afficher un message à l'utilisateur
Attention: l'utilisation de cette boîte de dialogue bloque toute exécution Javascript (y compris les gestionnaires d'événement) tant que l'utilisateur n'a pas cliqué sur OK ou CANCEL

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>exemple javascript</title>
7  <script>
8      function hello()
9      {
10          alert("Hello world !");
11      }
12  </script>
13 </head>
14 <body>
15     <p><a href="javascript:void hello()">hello</a></p>
16 </body>
17 </html>
```

Exemple: la méthode Window.confirm()

Cette méthode permet d'afficher un message dans une boîte de dialogue (pop-up) afin de demander une confirmation à l'utilisateur. Le message va s'afficher avec un bouton OK et un bouton CANCEL permettant à l'utilisateur de fermer ce pop-up à l'aide d'un des deux boutons

ok = confirm(message)

- Message est le message à afficher (généralement une question à poser)
- la valeur de retour est un booléen indiquant que l'utilisateur a cliqué sur OK (true) ou CANCEL (false)

Attention: l'utilisation de cette boîte de dialogue bloque toute exécution Javascript (y compris les gestionnaires d'événement) tant que l'utilisateur n'a pas cliqué sur OK ou CANCEL

```
7   <script>
8     function action()
9     {
10       var ok = confirm("Etes-vous sûr ?");
11       if (ok)
12       {
13         alert("allons-y !");
14         return true;
15       }
16       else
17       {
18         alert("abandon");
19         return false;
20       }
21     }
22   </script>
23 </head>
24 <body>
25   <p><a href="http://www.lesoir.be" onclick="return action()">action</a></p>
26 </body>
```

Exemple: la méthode Window.prompt()

Cette méthode permet d'afficher un message dans une boîte de dialogue (pop-up) afin de demander à l'utilisateur d'entrer une valeur. Le message va s'afficher avec un bouton OK et un bouton CANCEL permettant à l'utilisateur de fermer ce pop-up à l'aide d'un des deux boutons

valeur = prompt(message, défaut)

•*message* est le message à afficher (généralement une question à poser)

•*défaut* est la valeur par défaut à afficher dans la zone de saisie de texte

•la valeur de retour contiendra la réponse de l'utilisateur sous la forme d'une chaîne de caractères s'il a cliqué sur OK ou null s'il a cliqué sur CANCEL

Attention: l'utilisation de cette boîte de dialogue bloque toute exécution Javascript (y compris les gestionnaires d'événement) tant que l'utilisateur n'a pas cliqué sur OK ou CANCEL

```
7  <script>
8      function action()
9      {
10         let age = prompt("Quel est votre âge ?", "");
11         if (age==null) return false;
12         age = parseInt(age);
13         if ([age >= 18])
14         {
15             alert("allons-y !");
16             return true;
17         }
18         else
19         {
20             alert("abandon");
21             return false;
22         }
23     }
24 </script>
25 </head>
26 <body>
27     <p><a href="http://www.lesoir.be" onclick="return action()">action</a></p>
28 </body>
```

L'objet document

L'objet document est une propriété de l'objet window

On peut écrire `window.document`, mais comme `window` représente l'environnement global, on peut également écrire simplement `document` (le `window.` est implicite)

`document` représente la page html chargée dans le navigateur. C'est un objet du type **HtmlDocument** qui dérive du type **Document**

Si cette page change, l'objet document change. Inversement, si vous modifiez en Javascript le contenu de l'objet document, la page html affichée sera également modifiée

Tout changement effectué à l'objet document modifie dynamiquement la page affichée dans le navigateur

Voir `exemple03.html`

Les propriétés principales de l'objet document

.body	l'objet qui représente l'élément <body> de la page html
.head	l'objet qui représente l'élément <head> de la page html
.title	le titre de la page courante donné par l'élément <title>
.cookie	permet d'obtenir et de modifier les cookies
.domain	donne le domaine d'où provient la page courante (valeur importante pour les aspects de sécurité)
.forms	collection de tous les formulaires (éléments <form>) présents dans la page courante
.images	collection de toutes les images (éléments) présentes dans la page courante
.links	collection de tous les liens (éléments <a> et <area>) présents dans la page courante
<i>propriétés de DOM</i>	toutes les propriétés d'un objet Document et Node de DOM
...	

.write(...) .writeln(...)	permet d'écrire en Javascript le contenu ou une partie du contenu de la page html courante si celle-ci n'est pas encore totalement chargée
.close()	termine le chargement de la page courante
<i>méthodes de DOM</i>	toutes les méthodes d'un objet Document et Node de DOM

L'objet document

Modifier le contenu de la page html avec `document.write(...)`

Le code Javascript contenu dans une balise <script> peut écrire du code html dans la page en utilisant la méthode `document.write(...)` ou `document.writeln(...)`

Cette méthode, relativement harchaïque, n'est pas la meilleure méthode pour écrire dans la page html. Il faut lui préférer les méthodes qui utilisent [.innerHTML](#) ou [DOM](#)

Toutefois, on l'utilisera dans de nombreux exemples pour sa simplicité et sa compacité

Voir `exemple04.html`

L'objet document

Modifier le contenu de la page html avec innerHTML

Il est possible de modifier le contenu d'une balise html en utilisant les propriétés .innerHTML et .outerHTML de l'objet qui représente cette balise

Cet objet peut être retrouvé grâce à la méthode **document.getElementById(...)**

Voir exemple05.html

Modifier le contenu de la page html avec DOM

La façon la plus propre de modifier le contenu de la page est d'effectuer les modifications à l'aide de DOM
DOM ([Document Object Model](#)) est un interface de programmation qui modélise la page html chargée dans le navigateur à l'aide d'objets

Voir exemple06.html

pratique 02 - Savoir inclure du Javascript dans une page

1) Créez une page html avec une balise <script> au niveau du <head> de la page. Le code contenu dans cette balise doit créer deux variables destinées à contenir votre nom et votre prénom.

Utilisez ensuite la méthode document.write(...) dans le <body> de la page afin d'écrire une balise

<p>Bonjour prénom nom,</p>

où le nom et le prénom proviennent des deux variables ci-dessus.

2) Créez un fichier javascript séparé que vous chargerez dans le <head> de la page.

Le code contenu dans ce fichier doit définir une fonction destinée à écrire un

<p>Nous sommes le XX/XX/XXXX.</p>

à l'aide de document.write(). La date du jour est construite à l'aide d'un objet de type Date() (basez vous sur les exemples précédents et recherchez sur Internet les informations nécessaires pour obtenir le jour, le mois et l'année).

Appelez ensuite cette fonction dans une balise <script> que vous placerez après le <p> créé dans le 1er exercice.

3) Réalisez le même exercice en écrivant dans la propriété .innerHTML de l'objet représentant une

<div id="..."></div> placée dans le <body> de la page.

La fonction doit être appelée via le gestionnaire d'événement onload="..." (à inclure sur la balise body).

4) Réalisez le même exercice en DOM à l'aide de la méthode .appendChild(...).

03. Le DOM

DOM est un interface standardisé utilisé pour manipuler des documents xml/html à partir d'un langage de programmation orienté objets. DOM est multi-plateforme et multi-langage. DOM est bien entendu supporté par le langage Javascript

DOM modélise un document à l'aide d'une arborescence de **noeuds**. Chaque noeud représente une information trouvée dans le document xml/html (un élément, un attribut, un texte, etc.)

En programmation, chaque noeud sera matérialisé par un objet d'un certain type. Les types les plus importants sont:

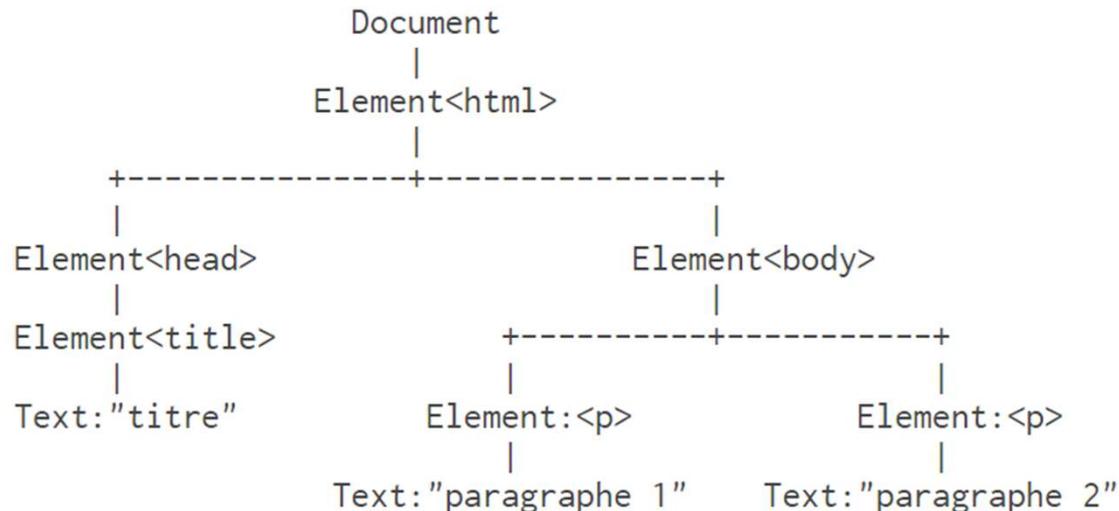
Document	pour modéliser le document lui-même
Element	pour modéliser un élément (une balise)
Attr	pour modéliser un attribut
Text	pour modéliser le texte contenu dans un élément (une balise)
...	

Tous ces types objets dérivent d'un type commun, le type **Node**. Ce dernier est utilisé pour représenter n'importe quel noeud trouvé dans un document (noeuds documents, noeuds éléments, noeuds attributs, noeuds textes, noeuds commentaires...)

Remarque: l'objet **document** que nous avons déjà vu est un objet du type **HtmlDocument** qui dérive du type **Document**. Il représente - ou modélise - le document html chargé dans le navigateur

L'arbre document du DOM

Les nœuds sont liés les uns aux autres via des relations de type **parent/enfants**



Le document modélisé par cet arbre est le suivant ->

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title>titre</title>
5  |  </head>
6  |  <body>
7  |  |  <p>paragraphe 1</p>
8  |  |  <p>paragraphe 2</p>
9  |  </body>
10 </html>
11 |
```

Les propriétés / Méthodes les plus importantes de Node

Un noeud modélisé par un objet **Node** possède les propriétés suivantes:

.nodeType	le type de noeud (1 pour les éléments, 2 pour les attributs, 3 pour les textes, 9 pour les documents...)
.nodeName	le nom associé au noeud (nom de l'élément en majuscule, nom de l'attribut en majuscule, "#document", "#text"...)
.nodeValue	la valeur associée au noeud (valeur de l'attribut, valeur du noeud texte, <code>null</code> pour les éléments et les documents...)
.parentNode	le noeud parent
.childNodes	la liste des enfants donnée par un objet de type NodeList . Possèdera la propriété <code>childNodes.length</code> et la méthode <code>childNodes.item(i)</code>
.firstChild	le premier noeud enfant. Idem que <code>.childNodes.item(0)</code>
.lastChild	le dernier noeud enfant. Idem que <code>.childNodes.item(childNodes.length - 1)</code>
.nextSibling	le noeud qui suit et qui a le même parent
.previousSibling	le noeud qui précède et qui a le même parent
.ownerDocument	le noeud Document qui contient ce noeud-ci (la racine de l'arborescence de noeud)
.hasChildNodes()	true si le noeud possède des enfants
.appendChild(new)	ajoute un noeud <code>new</code> à la fin de la liste des enfants (retourne le nouveau noeud ajouté)
.insertBefore(new, ref)	ajoute un noeud <code>new</code> dans la liste des enfants, juste avant le noeud <code>ref</code> (retourne le nouveau noeud ajouté)
.removeChild(old)	efface le noeud <code>old</code> (retourne le noeud effacé)
.replaceChild(new, old)	remplace le noeud enfant <code>old</code> par le nouveau noeud <code>new</code> (retourne le noeud effacé)
.cloneNode(deep)	crée un clone du noeud courant (ainsi que tous ses descendants si <code>deep</code> est égal à true)

Méthode

Les propriétés / Méthodes les plus importantes de Document

En DOM il n'y a pas beaucoup plus de propriétés, par contre l'objet **document** en Javascript, qui est du type **HtmlDocument** (un dérivé de **Document**), possèdera toute une série de propriétés supplémentaires:

.body	le noeud élément <body> du document
.forms	collection de tous les éléments <form> pour les formulaires
.images	collection de tous les éléments pour les images
...	

Les Méthodes les plus importantes de Element

Un document modélisé par un objet **Document** possède les méthodes suivantes:

.getElementsByName(<i>nom</i>)	retourne la liste de tous les éléments < <i>nom</i> > trouvés dans le document. Possèdera la propriété <i>résultat.length</i> et la méthode <i>résultat.item(i)</i>
.getElementById(<i>id</i>)	retourne l'élément qui possède un attribut <i>id="id"</i>
.importNode(<i>node</i> , <i>deep</i>)	importe le noeud <i>node</i> provenant d'un autre document dans le document courant (ainsi que tous ses descendants si <i>deep</i> est égal à true)
.createElement(<i>nom</i>)	crée un nouvel élément < <i>nom</i> >
.createTextNode(<i>texte</i>)	crée un nouveau noeud texte

Les propriétés / Méthodes les plus importantes de DOM

Exemple DOM: changer un contenu en fonction de la langue

Voir [exemple07.html](#)

Exemple DOM: modifier l'apparence du document

Voir [Exemple08.html](#)

Exemple DOM: autre façon de faire, en utilisant la propriété style

Par rapport à l'exemple précédent, il existe une autre manière de procéder en utilisant la propriété **style**.

Cette propriété **style** existe pour tous les objets qui représente une balise html. Elle est elle-même un objet, dont les propriétés représentent tous les propriétés CSS qui peuvent exister pour cette balise.

Le nom de ces propriétés est identique à celui des propriétés CSS (color, width, height...), toutefois, le '-' qui peut apparaître en CSS est supprimé et est remplacé par la lettre suivante mise en majuscule (fontFamily, fontSize, borderBottomColor, etc.).

Voir [exemple09.html](#)

pratique 03 : Augmenter / Diminuer la taille du texte

Créez une page web, contenant plusieurs paragraphes de texte en HTML (la plupart des graphistes et des développeurs web utilise pour cela le texte *Lorem Ipsum*).

Insérez dans cette page deux liens hypertextes, moins et plus, permettant de réduire ou d'augmenter la taille du texte à l'écran.

Pour cela, modifiez la propriété CSS font-size du <body> de la page, en sachant que l'objet qui représente cette balise est donné par la propriété **body** de l'objet **document**.

pratique 04 : Ecrire une horloge en DOM

Réalisez à l'aide de DOM une horloge

Cette page utilise une fonction changeHeure() qui s'appelle elle-même toutes les secondes. Cette fonction commence par retrouver l'élément qui est destiné à afficher l'heure. Elle efface ensuite le contenu de cet élément puis elle appelle la fonction afficheHeure() afin d'afficher la nouvelle heure

Cette fonction se trouve pour l'instant dans un fichier javascript séparé (inutile d'aller lire le code !). Vous pouvez remplacer le lien vers ce fichier séparé par un lien vers votre propre fichier, dans lequel vous créerez votre propre fonction afficheHeure().

Essayer de reproduire la même horloge, sans changer le code html et css de la page. Les instructions que vous allez ajouter dans la fonction afficheHeure() vont devoir remplir le contenu du à l'aide de DOM

Remarquez que les deux-points dans l'heure ont une couleur différente. Il faudra donc jouer avec des éléments supplémentaires et leur ajouter un attribut style="color:xxxx"

04. Please start your engine

Les variables... C'est quoi ?

Rappelez-vous pour certains lorsqu'ils étaient étudiants, l'école mettaient à leur disposition un casier numéroté. Dans le casier 35, les affaires de Jimmy étaient rangées. Le casier 35 occupe une place dans l'espace de l'école, et lorsque Jimmy veux accéder à ses affaire, il se rends à son casier et ouvre la porte.

C'est le principe des variables, c'est un espace mémoire que l'on va réservé dans le navigateur, qui aura un nom, pour qu'on puisse le retrouver facilement, et dans lequel on stockera de l'information (la valeur de la variable).



Déclarer une variable

3 types de variables:

VAR

Variable dont la valeur peut être modifiée en cours de programme

LET

Variable dont la valeur peut être modifiée en cours de programme

CONST

Variable dont la valeur restera identique tout au long du programme

```
var nomDeMaVariable = VALEUR;
```

```
let nomDeMaVariable = VALEUR;
```

```
const nomDeMaVariable = VALEUR;
```

Voir exemple10.js -> PARTIE1

Les mots réservés en Js

En Javascript, vous allez devoir choisir les noms de vos variables, de vos fonctions, de vos objets, de vos propriétés, de vos méthodes, de vos labels, etc.

Vous pourrez les choisir librement en commençant par une lettre (a à z, ou A à Z), un _ ou un \$, suivi d'autant de lettres (a à z, ou A à Z), de chiffres (0 à 9), de _ ou de \$ que vous voulez

Dans les versions récentes de Javascript, les lettres accentuées sont acceptées

Toutefois, vous ne pourrez utiliser aucun des **mots-clés réservés** du langage. Le langage utilise en effet certains noms pour ses propres besoins (le nom des instructions, par exemple). Ces noms lui sont réservé.

break – case – class – catch – const – continue – debugger – default – delete – do – else – export – extends – Finally – For – function – If – import – In – Instanceof – New – Return – Super – Switch – This – Throw – Try – Typeof – Var – Void – While – With – yield

Déclarer une variable

Les types de données primitifs

Toutes les données que l'on va manipuler en Javascript, ainsi que tous les résultats du calcul d'une expression, appartiendront toujours à un des **types de données primitifs** existant en Javascript

Ces **types primitifs** sont les suivants:

number	une valeur numérique. En réalité, un objet particulier qui possède toutes les caractéristiques du type Number sans en être un
string	une chaîne de caractères. En réalité, un objet particulier qui possède toutes les caractéristiques du type String sans en être un
boolean	une valeur booléenne. En réalité, un objet particulier qui possède toutes les caractéristiques du type Boolean sans en être un
object	un objet (n'importe quel type d'objet: String , Number , Array , Document , Element ...)
function	une fonction. En réalité, un objet particulier qui possède toutes les caractéristiques du type Function sans en être un
undefined	la valeur undefined

L'[**opérateur typeof**](#) est très pratique pour tester le type d'une valeur: il renverra une chaîne de caractères égal à "number", "string", "boolean", "object", "function" ou "undefined" (la valeur null renverra "object")

Voir exemple10.js -> PARTIE2

La concaténation

La concaténation va nous permettre d'ajouter des variables les unes aux autres pour afficher le résultat de cette concaténation.

```
1 const prénom = "Guy";
2 const nom = 'Vilain';
3
4 let salutation = "Bonjour " + prénom + " " + nom;
5
6 console.log(salutation);
7
8 const rencontre = "Je suis heureux de te rencontrer";
9
10 console.log(` ${salutation}, ${rencontre}`);
```

On peut insérer dans une constante de type chaîne de caractères les caractères spéciaux suivants:

- > \b: backspace (retour en arrière)
- > \f: form feed (saut de page)
- > \n: newline (saut de ligne)
- > \r: carriage return (retour chariot)
- > \t: tabulation
- > \\": backslash
- > \'': apostrophe
- > \"': guillemet
- > \uXXXX: caractère unicode XXXX (en hexadécimal)

Voir exemple11.js -> PARTIE1

Les opérateurs

Les cinq opérateurs arithmétiques de base qui s'appliquent à deux opérandes:

<i>opérande + opérande</i>	addition de deux opérandes numériques (sauf si une des deux est une chaîne de caractères, alors le + représente l' opérateur de concaténation)
<i>opérande - opérande</i>	soustraction de deux opérandes numériques
<i>opérande * opérande</i>	multiplication de deux opérandes numériques
<i>opérande / opérande</i>	division de deux opérandes numériques
<i>opérande % opérande</i>	modulo de deux opérandes numériques (reste de la division entière)

Ces opérateurs ne s'applique qu'à une seule opérande, qui doit être une [variable](#) (ou une propriété). Ils se placent avant ou après cette variable, généralement au sein d'une expression

<i>++variable</i>	incrémentation avant
<i>variable++</i>	incrémentation après
<i>--variable</i>	décrémentation avant
<i>variable--</i>	décrémentation après

Voir exemple11.js -> PARTIE2

Les boites de dialogue

3 types de boites:

ALERT()

La méthode `alert()` permet d'afficher dans une boîte toute simple composée d'une fenêtre et d'un bouton *OK* le texte qu'on lui fournit en paramètre. Dès que cette boîte est affichée, l'utilisateur n'a d'autre alternative que de cliquer sur le bouton *OK*.

Son unique paramètre est une chaîne de caractère, on peut donc lui fournir directement cette chaîne de caractères entre guillemets, lui fournir une variable dont il affichera le contenu, ou bien mélanger les deux en concaténant les chaînes grâce à l'opérateur `+`.

PROMPT()

La méthode `prompt` est un peu plus évoluée que les deux précédentes puisqu'elle fournit un moyen simple de récupérer une information provenant de l'utilisateur, on parle alors de boîte d'invite. La méthode `prompt()` requiert deux arguments :

- le texte d'invite
- la chaîne de caractères par défaut dans le champ de saisie

CONFIRM()

La méthode `confirm()` est similaire à la méthode `alert()`, si ce n'est qu'elle permet un choix entre "OK" et "Annuler". Lorsque l'utilisateur appuie sur "OK" la méthode renvoie la valeur `true`. Elle renvoie `false` dans le cas contraire...

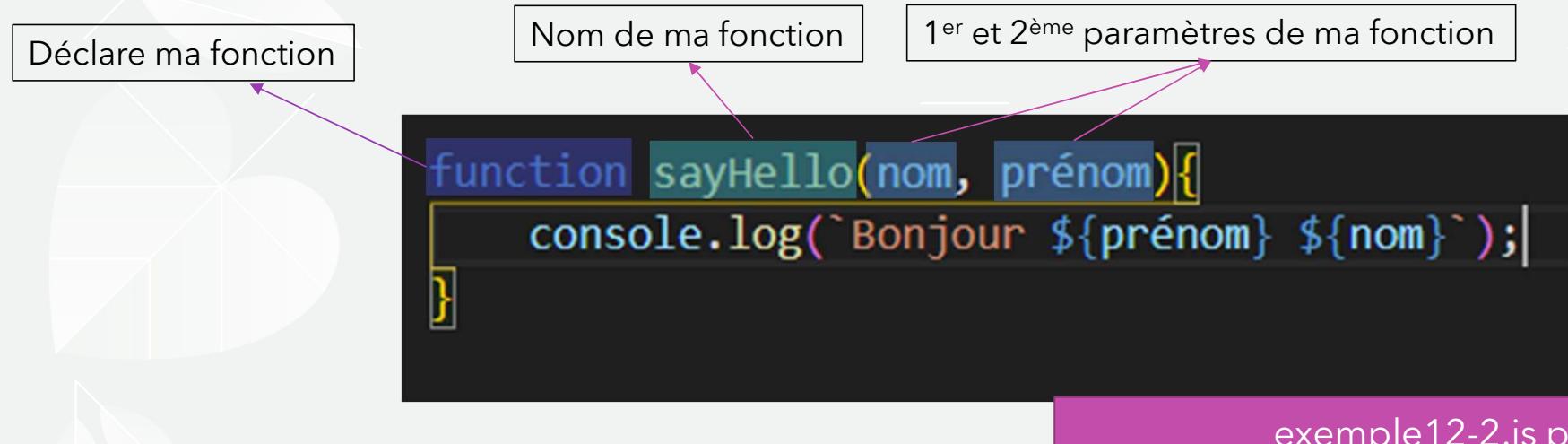
Voir exemple12.html et exemple12.js partie1

Les fonctions

Une fonction permet de regrouper un ensemble d'instructions destinées à une tâche précise (par exemple, calculer une valeur, afficher un menu, vérifier le contenu d'un formulaire...)

Une fonction doit au préalable être définie ou déclarée. La **déclaration de fonction** va, en général, déclarer le **nom de la fonction**, la liste des **paramètres**(si elle en possède) ainsi que le **corps de la fonction**, c'est-à dire la liste des instructions que la fonction va utiliser pour accomplir sa tâche

On pourra ensuite **appeler** la fonction en mentionnant son nom et en fournissant la liste des valeurs à donner aux paramètres. Cet appel déclenchera les instructions contenues dans le corps de la fonction



exemple12-2.js partie 2bis

Appel de ma fonction

```
9 <body>  
10   <button onclick="sayHello('Vilain', 'Guy')>Dire Bonjour</button>  
11 </body>
```

Les fonctions qui retourne quelques chose

Il faut faire une distinction entre les fonctions qui vont afficher une information, et une fonction qui va retourner une valeur.

```
49 ↴ function addition(n1, n2){  
50   |   let resultat = n1 + n2;  
51   |   alert(` ${n1} + ${n2} = ${resultat}`);  
52 }  
53
```

La fonction ci-dessus, nous sert à afficher le résultat d'une opération dans une boite de dialogue. Une fois la fonction effectuée, le résultat se perd dans la nature, je ne pourrai pas utiliser ce résultat à un autre endroit de mon programme.

```
93 ↴ let multiplication = function(n1, n2){  
94   |   return n1 * n2;  
95 }  
96  
97 console.log("j'ajoute 10 à la multiplication 20 x 2 = " + (multiplication(20, 2 ) + 10));  
98
```

La fonction ci-dessus demande de retourner le résultat d'une opération. Je suis maintenant capable d'utiliser ce résultat dans le reste de mon code.

[Voir exemple12.html et exemple12.js partie 3](#)

Les paramètres par défaut d'une fonction

Nous avons la possibilité de décider d'un comportement par défaut d'un paramètre d'une fonction. J'assignerai la valeur par défaut de ce paramètre dans la déclaration de la fonction (voir ci-dessous)

```
73  function diviserParDix(n1, n2 = 10){  
74    return n1 / n2;  
75  }  
76  
77  console.log(diviserParDix(100));  
78  console.log(diviserParDix(100, 11));
```

Lorsque j'utilise cette fonction, je dois seulement indiquer la paramètre qui n'a pas de valeur par défaut. Si plus tard dans le code, la valeur par défaut n'est plus pertinente, je peux lui réassigner une nouvelle valeur lorsque j'appelle ma fonction.

Voir exemple12.html et exemple12.js partie 4

La portée des variables

Voir exemple12.html et exemple12.js partie 5

Convertir des données

A de nombreuses reprises, dans un programme, tu vas recevoir des données qui ne seront peut-être pas dans le format dont tu as besoin. Heureusement, Js comprends une multitude de méthodes natives que tu pourras utiliser pour convertir des données d'un format dans un autre.

Voir exemple13.html et exemple13.js partie 1

Les fonctions anonymes

Les fonctions anonymes sont des fonctions qui n'ont pas de nom!

```
1  function(){}
2  |
3  |     console.log('je suis une fonction anonyme');
4  }
```

On peut invoquer une fonction anonyme de 3 manières:

- On la stocke dans une variable
- En l'exécutant immédiatement (lorsqu'on clique sur un bouton par exemple)
- En l'utilisant dans un évènement (voir plus tard dans le cours)

Les fonctions anonymes

Fonction anonyme dans une variable:

```
6  let direBonjour = function (){  
7      console.log("Bonjour toi!");  
8  }
```

Pour l'utiliser

```
10  direBonjour();|
```

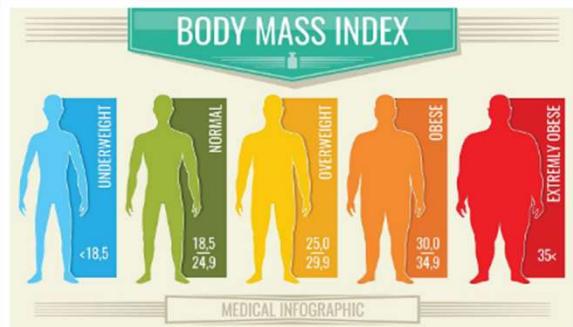
Fonction anonyme seule:

```
12  function(){  
13      console.log("Je ne suis pas opérationnelle de cette manière");  
14  }|
```

Pour l'utiliser, je dois l'auto-exécuter en l'écrivant sur une ligne, et en l'englobant dans des ()

```
12  (function(){console.log("Je suis pas opérationnelle de cette manière")})();|
```

Voir exemple13.html et exemple13.js partie 2



EXERCICE 01 – Calcul de l'IMC

Dans cet exercice, je vous propose de réutiliser tout ce que nous avons vu jusqu'à maintenant. Si vous le réussissez, vous pourrez définitivement valider toutes les notions que nous avons déjà vu ensemble !

Voici ce que nous allons faire : **un calculateur d'IMC** !

- Nous allons récupérer plusieurs valeurs grâce à notre utilisateur : nom, prénom, rue, le n°d'habitation, la localité, le code postal, l'âge, l'urlphoto le **poids** et la **taille**, qui seront respectivement associées au poids et à la taille de notre utilisateur. Vous pouvez demander à vos utilisateurs leur taille en **centimètres** ou en **mètres**. Dans tous les cas, vous devrez convertir cette taille en **mètres** pour calculer son IMC.
- Il faudra ensuite passer ces valeurs à notre fonction, **grâce à ses paramètres**. J'insiste sur ce point.
- Dans cette fonction **calculerIMC** nous aurons une formule mathématique, que je vais vous donner car il n'y a pas d'intérêt à la chercher :

$$\frac{\text{poids (kg)}}{\text{taille}^2 (\text{m})}$$

Notez bien que le poids doit être en kg, et la taille en mètres. Donc, par exemple : 53kg et 1.50m.

Enfin, l'objectif pour notre fonction sera de nous **retourner** ce résultat afin que nous puissions l'afficher à notre utilisateur, directement via une **boîte de dialogue en dehors** de notre fonction. Se limiter à deux chiffres après la virgule!

Les conditions en Javascript

Pour autoriser l'accès à mon site aux personnes majeure, je dois pouvoir vérifier leur âge. Je vais donc conditionner l'accès de mon site à l'âge de l'utilisateur. Js nous permet de gérer les conditions grâce à différentes instructions conditionnelles.

On commence à entrer dans la logique de js, accroche toi, c'est parti!!!

L'instruction conditionnelle if

L'instruction conditionnelle s'écrit de la façon suivante:

```
1  if(maCondition) mon instruction
2
3  if(maCondition) Mon instruction else Mon instruction contraire
```

- La clause else n'est pas obligatoire
- La condition peut être n'importe quelle expression (généralement il s'agira d'une expression qui retourne une valeur booléenne) (exemple si a = b)
- La vérification de la condition nous retourne un boolean. Si c'est true exécute la première instruction, sinon exécute la deuxième instruction si le else est présent sinon ne fait rien
- Si plusieurs instructions doivent être exécutées, il faut utiliser une [instruction de bloc](#) (plusieurs instructions entre { et })

```
5  if(maCondition){
6      // Instruction 1
7      // Instruction 2
8      // Instruction 3
9  } else {
10     // Instruction 1
11     // Instruction 2
12     // Instruction 3
13 }
```

Les conditions en Javascript

Un instruction conditionnelle peut sans problème exécuter une autre instruction conditionnelle en fonction de la condition
La construction suivante est très souvent utilisé (le premier if exécute un else qui contient un deuxième if et ainsi de suite)

L'instruction conditionnelle s'écrit de la façon suivante:

```
15 let color;
16
17 ↘ if (couleur=='rouge')
18 {
19   color='red';
20   code="#FF0000";
21 }
22 ↘ else if (couleur=='vert')
23 {
24   color='green';
25   code="#00FF00";
26 }
27 ↘ else if (couleur=='bleu')
28 {
29   color='blue';
30   code="#0000FF";
31 }
32 ↘ else
33 [
34   color='black';
35   couleur = "noir";
36   code="#000000";
37 ]
```

Les conditions en Javascript

Les opérateurs de comparaison.

Pour comparer deux valeurs entre elles, j'ai besoin des opérateurs de comparaison.

Voici ceux dont nous disposons en js (et en définitive dans tous les langages de programmation)

- `==` : égal à la valeur
- `====` : égal à la valeur et au type
- `!=` : différent de la valeur
- `!==` : différent de la valeur et du type
- `>` : supérieur
- `<` : inférieur
- `>=` : supérieur ou égal
- `<=` : inférieur ou égal

Vérifier plusieurs conditions en une fois!!!

Je peux vérifier plusieurs conditions en une fois grâce aux opérateurs OR (`||`) et AND (`&&`)

[Voir exemple14.js Partie 1](#)

Les conditions en Javascript

L'instruction de sélection switch

Une instruction switch évalue une expression (le calcul est effectué une seule fois) et compare le résultat avec les différentes valeurs proposées. La première valeur strictement égale déclenche les instructions qui la suivent

Une instruction break permet d'interrompre cette exécution (et passer à la suite du switch). Si elle n'est pas présente, l'exécution continuera avec les instructions du case qui suit

Si aucune valeur ne correspond, les instructions du default éventuel seront exécutées (il est généralement à la fin, mais ce n'est pas une obligation)

```
70  switch (couleur)
71  {
72    case 1: color="red";
73    |         break;
74    case 2: color="blue";
75    |         break;
76    case 3:
77    case 4: color="green";
78    |         break;
79    default: alert("couleur inconnue");
80    case 5: color="black";
81    |         break;
82 }
```

Voir exemple14.js Partie 2

Les boucles en Javascript

L'instruction de boucle while

L'instruction while est une instruction de boucle qui répète une instruction tant qu'une condition est évaluée à false ou true. Pour clore cette boucle(et toutes les autres), il est donc important d'y glisser un paramètre qui permettra de sortir de cette boucle, sans cela, la boucle se répète indéfiniment et peut faire au mieux planter votre navigateur, au pire votre ordinateur.

```
1 let i = 1;
2
3 while(i <= 10){
4   console.log(`J'execute un jeu d'instruction`);
5 }
```

```
1 let i = 1;
2
3 while(i <= 10){
4   console.log(`J'execute un jeu d'instruction`);
5   i++;
6 }
```

La boucle while ne s'exécutera pas si la condition de sortie est déjà remplie.

```
1 let i = 10;
2
3 ↘ while(i < 10){
4   console.log(`J'execute un jeu d'instruction`);
5   i++;
6 }
7
```

Voir exemple15.js Partie 1

Les boucles en Javascript

L'instruction de boucle Do ... while

L'instruction do while fonctionne sur le même principe que la précédente à la différence que le bloc d'instruction à exécuter s'executera une fois, même si la condition de sortie est déjà remplie:

```
18  ****
19  *****PARTIE 2*****
20  ****
21
22 // la boucle Do While
23
24 let z = 10;
25
26 do{
27     console.log(` j'écris la boucle do... While ${z}`);
28     i++;
29 }while(z<10); //Va écrire la ligne même si la condition de sortie est déjà remplie
```

Voir exemple15.js Partie 2

Les boucles en Javascript

L'instruction de boucle for...

L'instruction de boucle for exécute une première expression généralement destinée à initialiser une variable qui va servir d'index. Elle répète ensuite une instruction tant qu'une condition est évaluée à true, tout en exécutant une deuxième expression à la fin de chaque boucle. Cette deuxième expression est généralement destinée à faire varier la variable qui sert d'index.

```
for (expression-initiale; condition; expression-itérative) instruction
```

```
35 | for(i=0; i<=10; i++){  
36 |   console.log("J'écris la ligne " + i + ".");  
37 }
```

Un instruction [break](#) permettra d'arrêter la boucle et une instruction [continue](#) permettra de recommencer la boucle(ceci est valable pour toutes les boucles)

Voir exemple15.js Partie 3

Il existe d'autres boucles, [for ...in](#), [for ... of](#), [foreach](#), qui sont utilisée pour parcourir des objets → [Cours js avancés](#)

Try ... Catch

L'instruction try catch permet d'exécuter des instructions comme dans une [instruction de bloc](#) { }

Si l'une d'elles génère une exception (une erreur), cette exception sera capturée par la clause catch et les instructions que cette clause contient seront exécutées.

```
try {
    instruction; instruction; instruction;...
}
catch (variable) // optionnel si finally est présent
{
    instruction; instruction; instruction;...
}
finally          // optionnel si catch est présent
{
    instruction; instruction; instruction;...
}
```

- L'instruction doit avoir une clause catch ou une clause finally ou les deux
- L'instruction exécute les instructions qu'elle contient entre les {} (comme dans une [instruction de bloc](#))
- Si une exception est générée (soit via une erreur, soit via l'instruction [throw](#)) la valeur de cette exception (généralement un objet décrivant l'erreur ou le résultat de l'expression donnée dans le [throw](#)) sera affectée à la variable mentionnée, et les instructions définies dans le catch seront exécutées
- Si aucune exception n'est générée, la clause catch sera ignorée
- Dans tous les cas (qu'une exception soit générée ou non), les instructions de la clause finally éventuelle seront exécutées
- L'exception ne sera pas transmise au reste du programme, car elle a été capturée par la clause catch ou par la clause finally

Voir exemple15.js Partie 4

Exercice 02: Coder une calculatrice

Réaliser une mini calculatrice pouvant effectuer les 4 opérations de base, à savoir Addition / Soustraction / Multiplication / Division. L'opération et les deux nombres à opérer seront donnés par l'utilisateur lorsqu'il lancera votre programme.

On commence par demander à l'utilisateur quel type d'opération il veut effectuer.

L'utilisateur choisit le premier nombre

L'utilisateur choisit le second nombre

Le programme effectue l'opération choisie par le user et affiche la réponse.

Si l'utilisateur demande une opération qui n'existe pas, reposer la question jusqu'à ce qu'il choisisse une opération disponible (de 1 à 4)

Pensez à utiliser une vérification sur la division, si le user veut diviser par 0, un message d'erreur doit apparaître.

Une fois le résultat de l'opération affichée, proposer à l'utilisateur un nouveau calcul. Si oui, relancer la calculatrice. Si non, remercier le user et quitter le programme.

On peut concevoir ce programme en 6 étapes que nous listerons sur le slide suivant:

Voir correction
exercices02.js

ETAPE 1

Dans cette première étape, je vais vous demander d'afficher un menu à notre utilisateur, comportant les 4 modes que nous lui proposons :

- 1.Addition
- 2.Multiplication
- 3.Soustraction
- 4.Division

Nous utiliserons ce menu pour récupérer un nombre : 1, 2, 3 ou 4, qui nous permettra de savoir ce que l'utilisateur souhaite faire. Nous stockerons ce choix dans une variable **choix**.

ETAPE 5

Dans cette étape, il ne nous reste qu'une seule chose à faire : afficher le résultat. Je n'ai pas de conseil particulier ici, si ce n'est de bien penser à afficher le résultat dans une **boîte de dialogue** et le faire **après** votre switch. N'oubliez pas : nous ne voulons pas nous répéter dans notre code !

ETAPE 2

Dans cette deuxième étape, je vous invite à demander deux nombres à utiliser lors de vos calculs. Ces deux nombres seront stockés dans deux variables: **premierNombre** et **deuxiemeNombre**.

Attention : Vérifiez bien qu'il s'agit de deux nombres, nous ne sommes pas à l'abri d'un utilisateur qui rentrerait du texte !

Astuce : Pour vérifier si une valeur est bien un nombre, vous pouvez utiliser une boucle **do...while**, et utiliser la fonction **isNaN(variable)** qui renvoie *true* si la variable n'est pas un nombre.

ETAPE 3

Il est maintenant temps de créer nos 4 fonctions :

- addition**(nombreA, nombreB)
- multiplication**(nombreA, nombreB)
- soustraction**(nombreA, nombreB)
- division**(nombreA, nombreB)

Je ne vous aide pas sur cette étape, pensez juste à bien **retourner** le résultat, et non pas l'afficher.

ETAPE 4

Nous avons maintenant le mode souhaité par l'utilisateur, nos modes et deux nombres. Il ne nous reste plus qu'à appeler la bonne fonction. Ce que vous devez faire :

- Utiliser un **switch** pour vérifier plusieurs cas : 1, 2, 3 et 4
- Dans chaque cas, appeler la fonction souhaitée. Par exemple, si choix vaut 1, nous appelons la fonction addition. Stocker la valeur de retour de notre fonction dans une variable **resultat**.

ETAPE 6

Nous voici dans la dernière étape. Notre calculatrice fonctionne déjà très bien, toutefois, il faut que nous pensions aux erreurs qui peuvent être rencontrées. Pour ceci, nous allons créer des exceptions.

Je voudrai que vous créiez deux exceptions :

- Une dans le **default** de votre switch, disant qu'une erreur est survenue dans une alerte. Nous ne sommes jamais à l'abri d'un bug.
- Une dans la fonction **division()**, lorsque **nombreB** vaut **0**. En effet, il est interdit de diviser par 0 ! C'est une règle fondamentale des mathématiques ! Nous devons donc créer une exception "Impossible de diviser par 0.".

Les tableaux

Un tableau est une variable qui peut contenir non pas une, mais plusieurs valeurs. C'est une sorte de succession ordonnées de "cases" ou de **cellules** contenant chacune une valeur

Le constructeur `Array()` permet de créer un tableau qui est un objet de type `Array`.

```
1 //Via Constructeur
2 let couleurs = new Array("rouge", "bleu", "vert", "jaune");
3
4 //Via littéral tableau
5 let outils = ["scie", "marteau", "hache", "tronçonneuse", "tournevis"];
```

Utilisé sans arguments, crée un tableau vide en attente de donnée, sa longueur (`array.length`) sera égale à 0.

Je peux parcourir un tableau en utilisant la variable qui le contient et en demandant d'afficher un index précis, dans l'exemple ci-dessus:

```
4 //Via littéral tableau
5 let outils = ["scie", "marteau", "hache", "tronçonneuse", "tournevis"];
6
7 outils[0] //scie
8 outils[1] //marteau
9 outils[2] //hache
10 outils[3]//tronçonneuse
11 ...
12 ...
```

IMPORTANT: Le premier élément d'un tableau est rangé à l'index 0, le second à l'index 1 et ainsi de suite

[Voir exemple16.js partie1](#)

Les tableaux

Toutes les valeurs d'un tableau ne doivent pas être définies (il peut y avoir des cellules vides)

```
11 //Définir un tableau avec des cellules vides:  
12  
13 let langages = ["html", "", "javascript", "", "php", "sql", "", "Python"];  
14 console.log(langages.length); //8|
```

La longueur d'un tableau (ou sa dimension)

La propriété **.length** retourne la longueur d'un tableau

```
4 //Via littéral tableau  
5 let outils = ["scie", "marteau", "hache", "tronçonneuse", "tournevis"];  
6  
7 console.log(outils.length);//5  
8  
9 console.log(` le tableau contient ${outils.length} éléments. `);|
```

Voir exemple16.js partie2

Les tableaux

Les tableaux multidimensionnels

on peut stocker de nombreux types de valeurs dans les cellules de notre tableau, string, number, boolean, objet, et on peut aussi stocker d'autres tableaux!!! C'est ce qu'on appelle un tableau multidimensionnel.

Pour accéder à une valeur particulière de notre tableau, nous identifierons d'abord l'index du tableau où se trouve la valeur recherchée, et ensuite l'index de la valeur.

```
41 let monTableau2D = [
42   ['Mark', 'Jeff', 'Bill'],
43   ['Zuckerberg', 'Bezos', 'Gates']
44 ];
45
46 console.log(`la première valeurs de chaque tableau est ${monTableau2D[0][0]} ${monTableau2D[1][0]}.`);
47 console.log(`les secondes valeurs de chaque tableau est ${monTableau2D[0][1]} ${monTableau2D[1][1]}.`);
48 console.log(`les troisièmes valeurs de chaque tableau est ${monTableau2D[0][2]} ${monTableau2D[1][2]}.`);

TERMINAL CONSOLE DE DÉBOGAGE PROBLÈMES SORTIE
D:\programs\node\node.exe .\exemple16.js
> (6) ['janvier', 'février', 'mars', 'avril', 'mai', 'juin']
la première valeurs de chaque tableau est Mark Zuckerberg.
les secondes valeurs de chaque tableau est Jeff Bezos.
les troisièmes valeurs de chaque tableau est Bill Gates.
```

Voir exemple16.js Partie03

Les tableaux

Les tableaux associatifs

Tableaux finalement assez proche d'un objet. Le tableau ne contient plus de cellule à proprement parler, mais une combinaison de données clé / Valeur que l'on va pouvoir appeler à loisir, ce qui se révèlera dans beaucoup de projet bien plus pratique.

```
1 let guyVilain = {  
2   nom : "vilain",  
3   prénom : "Guy",  
4   rue : "Rue François Dive",  
5   numéro : 10,  
6   codePostal : 5060,  
7   Localité : "Falisolle",  
8   section : "développement web",  
9 }  
10 |
```

Voir exemple16.js -
Partie 4

Pour lire une donnée de ce tableau, je fais appel à la variable qui le contient suivi d'un .nomDeLaClé

```
10  
11 console.log(` je vous présente Monsieur ${guyVilain.nom} ${guyVilain.prénom},\nrésidant à la ${guyVilain.rue}, N°${guyVilain.numéro}\`|  
12 \n${guyVilain.codePostal} ${guyVilain.Localité}\nMonsieur ${guyVilain.nom} à choisi l'option ${guyVilain.section}`);
```

TERMINAL CONSOLE DE DÉBOGAGE PROBLÈMES SORTIE Filtre (exemple : text, !exclude)

```
D:\programs\node\node.exe .\exemple17.js  
je vous présente Monsieur Vilain Guy,  
résidant à la Rue François Dive, N°10  
5060 Falisolle  
Monsieur Vilain à choisi l'option développement web
```

Les tableaux

L'ajout d'une cellule peut se faire à tout moment, soit en utilisant la propriétés .length :

```
13 let langages = ["html", "", "javascript", "", "php", "Sql", "", "Python"];
14 // console.log(langages.length); //8
15
16 //Ajouter une cellule avec la propriété .length
17
18 langages[langages.length] = "basic";
19 langages[langages.length] = "Pascal";
20 console.log(langages);
```

TERMINAL CONSOLE DE DÉBOGAGE PROBLÈMES SORTIE
D:\programs\node\node.exe .\exemple16.js
> (10) ['html', '', 'javascript', '', 'php', 'Sql', '', 'Python', 'basic', 'Pascal']

Soit en utilisant la méthode .push():

```
21 //Ajouter une cellule avec la méthode push()
22 langages.push("Ruby");
23 langages.push("Java");
24 langages.push("Swift");
25 console.log(langages);
```

TERMINAL CONSOLE DE DÉBOGAGE PROBLÈMES SORTIE
D:\programs\node\node.exe .\exemple16.js
> (13) ['html', '', 'javascript', '', 'php', 'Sql', '', 'Python', 'basic', 'Pascal', 'Ruby', 'Java', 'Swift']

Les tableaux

Une ou plusieurs cellules peuvent être supprimées à tout moment, grâce à la méthode `.splice(...)`:

```
26 //Supprimer une cellule avec splice
27 langages.splice(1,1);
28 langages.splice(2,1);
29 langages.splice(4,1);
30 console.log(langages);
```

TERMINAL CONSOLE DE DÉBOGAGE PROBLÈMES SORTIE
D:\programs\node\node.exe .\exemple16.js
> (10) ['html', 'javascript', 'php', 'sql', 'Python', 'basic', 'Pascal', 'Ruby', 'Java', 'Swift']

Le premier paramètre est l'indice de la première cellule à supprimer et le deuxième paramètre est le nombre de cellules à supprimer (en partant de l'indice)

Cette méthode offre également la possibilité d'ajouter des nouvelles cellules ou de remplacer des cellules par une autre valeur.

```
31 //Ajouter une cellule avec splice
32
33 let mois = ["janvier", "mars", "mai", "juillet"];
34 mois.splice(1, 0, "février");//Ajoute février à l'indice 1
35 mois.splice(3, 0, "avril"); //Ajoute avril à l'indice 3
36 mois.splice(5, 1, "juin");//Remplace juillet par juin
37 console.log(mois);
```

Les tableaux

Voici un résumé des principales fonctionnalités liées aux tableaux, que nous avons vu jusqu'à présent.

Pour ces exemples, nous prendrons en compte le fait que nous avons ce tableau :

```
let fruits = ['pomme', 'banane', 'poire', 'fraise']
```

Ainsi, voici les principales fonctionnalités :

- `fruits.length` : retourne le nombre d'éléments dans le tableau (ici retourne 4)
- `fruits[0]` : sélectionne le premier élément
- `fruits[length - 1]` : sélectionne le dernier élément
- `fruits.push('pamplemousse')` : ajoute un élément à la fin du tableau
- `fruits.unshift('pamplemousse')` : ajoute un élément au début du tableau
- `fruits.pop()` : supprime le dernier élément
- `fruits.shift()` : supprime le premier élément
- `fruits.indexOf('banane')` : retourne l'index d'un élément
- `fruits.join()` : concatène les éléments dans une chaîne de caractères avec virgules, mais il est possible de spécifier un autre séparateur dans les parenthèses
- `fruits.slice()` : crée une copie du tableau (à associer à une autre variable donc)
- `fruits.splice([début], [nbASupprimer], [élément(s)])` : retire, remplace ou ajoute des éléments.
 - Début** : l'index à partir duquel commencer le changement, si négatif, part de la fin du tableau
 - nbASupprimer** : un entier indiquant le nombre d'éléments à retirer ou remplacer
 - Element(s)** : les éléments à ajouter à partir du début précisé. Si aucun élément n'est spécifié, alors n'en ajoutera pas.

Voir exemple17.js -
Partie 1

Les tableaux

Parcourir un tableau avec une boucle for...in

L'instruction for...in permet de parcourir simplement toutes les propriétés énumérables(qui peut être énumérée dans une boucle) d'un objet (dans ce cas, un tableau)

```
26  for (const key in object) {  
27  
28  }  
29  |
```

Une boucle implicite va avoir lieu, dans laquelle la variable va recevoir successivement le nom de chaque propriété énumérable de l'objet (il faudra utiliser un accesseur de propriété ([« propriété »]) pour obtenir la valeur)

Voir exemple18.js - Partie 1

Parcourir un tableau avec une boucle for...of

Même combat que pour for..in, sauf qu'ici, pas besoin d'accesseur de propriété pour accéder à la valeur, puisque c'est elle qui est directement appelée:

Voir exemple18.js - Partie 2

Les tableaux

Parcourir un tableau avec une boucle foreach

Avec la boucle foreach, je vais pouvoir exécuter une fonction sur chaque élément du tableau:

```
70  ↴ array.forEach(element => {  
71      // Instructions  
72  });|
```

element sera la variable temporaire dans laquelle sera stockée chaque cellule du tableau à chaque itération de la boucle. La fonction (fléchée) sera exécutée à chaque itération.

[Voir exemple18.js - Partie 3](#)

Enfin tout comprendre du DOM

Nous allons revenir sur une notion que nous avons survolé en début de module et approfondir la notion de DOM(Document Object Model). Si tu te souviens bien, le dom est une cartographie de notre page html et nous permet de pouvoir cibler un élément précis (une balise, un attribut, du texte,...), dans le but de pouvoir y appliquer une modification.

Le moment est donc venu d'étudier sérieusement comment interagir avec le DOM, en lui ajoutant / supprimant des éléments, changer le style de ses composants, mais avant tout un rappel important:

Certains apprenants pourront tourner en rond pendant quelques heures à cause d'un « bug » : leur code JavaScript ne s'exécutera pas, alors qu'il sera bien connecté à leur page html...

La raison sera simple: la balise `<script src="code.js"></script>` sera placée dans leur balise `<head></head>`.

Pourquoi ce code ne permet pas d'exécuter du JavaScript ?

Parce que lorsque vous appelez votre fichier JavaScript, votre navigateur va le lire directement... alors qu'il n'est pas encore passé par la balise `<body></body>` (et donc, qu'il ne sait pas ce que contient votre page) !

En faisant comme ceci, lorsque vous voudrez sélectionner un élément de votre page, votre navigateur vous dira qu'il n'a pas trouvé l'élément. Heureusement, il y a une solution toute simple.

Mettez votre balise `<script></script>` juste avant la fin de votre `</body>`

En mettant votre balise `<script></script>` juste avant la fin de votre `</body>`, votre navigateur a eu le temps de lire l'ensemble de votre page, il sait parfaitement s'il y a des titres, combien il y a de liens, etc... Il peut donc trouver vos éléments sans problème grâce à JavaScript ! :-)

Accéder aux éléments du DOM

Sur cette page html, nous avons un header dans lequel j'ai une image contenue dans une div avec l'id « logo ». J'ai en dessous un titre h1 et deux paragraphes contenus dans une balise ayant la classe « container ».

Voir exemple19.html

Les méthodes pour accéder à ces éléments:

Pour utiliser les méthodes nous devons préciser l'endroit ou aller capturer ces éléments, cet endroit, c'est notre document que nous avons survolé en début de module, rappelle-toi que nous pouvons aussi insérer ces méthodes de capture dans des variables.
Voyons la première méthode:

`getElementsByTagName('nomDeLaBalise')` => nous permet de capturer toutes les balises en paramètres en vue de les modifier !

Si je veux capturer le header et les paragraphes de ma page html, je coderai:

```
1 let header = document.getElementsByTagName("header");
2
3 let paragraphe = document.getElementsByTagName("p");
4
5 console.log(header); //renvoie un tableau avec un élément: header
6 console.log(paragraphe); //renvoie un tableau avec deux éléments: deux balises p
7 |
```

Accéder aux éléments du DOM

Voir exemple19.js Partie 1

`getElementById('nomDeLId')` => nous permet de capturer la balise sur lequel l'id est posé pour la manipuler !

Si je veux capturer la div comprenant mon logo, je coderai:

```
8  let logo = document.getElementById('logo');
9  console.log(logo);
10 |
```

`getElementsByClassName('nomDeLaClass')` => nous permet de capturer la/les balises sur lesquelles les classes sont posées pour les manipuler !

Si je veux capturer la div avec la class container, je coderai:

```
11 let container = document.getElementsByClassName('container');
12 console.log(container);
```

`querySelector('elementACapturer') / querySelectorAll()` => nous permet indifféremment des balises, des id, et des class

```
1  let header = document.querySelector('header');//selectionner balise
2  let paragraphe = document.querySelectorAll('p');//selectionner toutes les balises p
3  let titleh1 = document.querySelector('h1');//selectionner balise
4  let logo = document.querySelector('#logo');//selectionner id
5  let container = document.querySelector('.container');//selectionner class|
```

Modifier les éléments du DOM

Voir exemple19.js Partie 2

Maintenant que nous savons identifier en javascript différents endroit de mon dom, je vais pouvoir les modifier à loisir avec plusieurs méthodes que nous allons partiellement lister:

textContent => va me permettre de poser ou de remplacer du texte dans la balise/class/id capturé

```
1 let h1 = document.querySelector('h1');
2 h1.textContent = "Hello World !!!"; //remplace le contenu de mon h1 par le texte|
```

innerHTML => nous cette fois d'injecter dans notre page du code html

```
1 let h1 = document.querySelector('h1');
2 h1.innerHTML = "<h2 style=\"color: red;\">Ceci est un nouveau titre</h2>";|
```

Les propriétés de modifications sont trop nombreuses que pour être énumérées durant cette initiation, mais comme toujours, la doc de js est votre amie:

<https://developer.mozilla.org/fr/docs/Web/API/Element>

Ajouter des éléments au DOM

Voir exemple19.js Partie 3

Il existe plusieurs méthodes pour ajouter des éléments au dom. Nous avons déjà vu la première dans l'introduction de ce module, qui nous permet d'écrire du texte à la fin de notre document, ce texte est injecté juste avant la fermeture de la balise body

`document.write()`

```
1  document.write("Je suis une nouvelle ligne de texte");|
```

Append() => Pour ajouter un élément brut (des chaînes de caractère) et choisir où je veux l'injecter en utilisant un selecteur. Le contenu sera injecté après le contenu déjà présent dans le selecteur.

```
1  let container = document.querySelector('h1');
2
3  container.append('j\'écris du nouveau texte que j\'injecte dans ma div container,\n'
4    'à la fin du contenu déjà existant');|
```

prepend() => Pour ajouter un élément brut (des chaînes de caractère) et choisir où je veux l'injecter en utilisant un selecteur. Le contenu sera injecté avant le contenu déjà présent dans le selecteur.

```
1  let container = document.querySelector('h1');
2
3  container.prepend('j\'écris du nouveau texte que j\'injecte dans ma div container,\n'
4    'à la fin du contenu déjà existant');|
```

Ajouter des éléments au DOM

Voir exemple19.js Partie 3

Je peux aussi créer un tout nouvel élément html que je pourrai injecter dans ma page, la création de ce nouvel élément ce fait en trois étapes:

- 1/ Créer l'élément => createElement()
- 2/ Personnaliser l'élément
- 3/L'injecter dans la page

```
2 //la variable HelloWorld va créer une nouvelle balise h3
3 let helloWorld = document.createElement('h3');
4
5 //J'injecte le texte que je désire ajouter à mon titre h3
6 helloWorld.textContent = "Hello le monde!!!";
7
8 //L'ajouter en bas de ma page:
9 document.body.append(helloWorld);
```

insertBefore(elementARajouter, endroitOuLeRajouter) => Autre méthode pour rajouter un éléments sur notre page html en visant un endroit précis. En premier paramètre, je place le contenu à injecter, en second paramètre, l'endroit où il sera injecté: (préférer prepend() qui est plus direct) :

```
1 let container = document.querySelector(".container");
2
3 let helloWorld = document.createElement('h3');
4 helloWorld.textContent = "Salut nounou !";
5
6 document.body.insertBefore(helloWorld, container);
7 |
```

supprimer des éléments au DOM

Voir exemple18.js Partie 4

Pour supprimer définitivement un élément de notre page html, on peut faire au choix:

```
1 let h1 = document.querySelector('h1');
2
3 h1.remove();|
```

Ou plus simplement

```
1 document.querySelector('h1').remove();|
```

Modifier le style des éléments au DOM

Voir exemple19.js Partie 5

Je peux modifier le style de mes éléments assez facilement, et je peux le faire de deux manières:

- ⇒ Propriété **style.nomDeLaPropriété**: le nom de la propriété équivaut à la propriété css que l'on veut appliquer sur notre élément. Attention néanmoins, les propriétés avec des noms composés, perdent leur tirets et s'écrivent en CamelCase. background-color deviendra backgroundColor par exemple.
- ⇒ Rajouter / retirer une classe avec la propriété **className('nomDeLaClass')**: je peux tout aussi bien code une classe en css dans ma feuille de style et décider de l'activer sur un élément ou non! **ATTENTION** la classe que je vais rajouter va écraser l'éventuelle classe déjà présente sur un élément.

```
1 //Avec la propriété style méthode décomposée
2
3 let header = document.querySelector('header');
4
5 header.style.backgroundColor = "red";
6 header.style.border = "2px solid black";
7
8 //Avec la propriété style méthode directe
9
10 document.querySelector('img').style.padding = '2rem';
11 document.querySelector('h1').style.textAlign = "center"
12
13 //Avec la propriété className
14
15 let container = document.querySelector('.container');
16
17 container.className = "modeNuit container";|
```

Le DOM en résumé

Voici une fiche technique résumant l'ensemble des fonctions et propriétés principales liées au DOM.

Accéder aux éléments

- getElementsByName()** - Sélectionne tous les éléments avec la balise entre parenthèses
- getElementById()** - Sélectionne un seul élément : le premier ayant l'ID entre parenthèses
- getElementsByClassName()** - Sélectionne tous les éléments avec la classe entre parenthèses
- querySelector()** - Sélectionne un seul élément : celui avec le sélecteur entre parenthèses
- querySelectorAll()** - Sélectionne tous les éléments avec le sélecteur entre parenthèses

Modifier les éléments

- textContent** - Modifie le texte d'un élément
- innerHTML** - Modifie l'HTML d'un élément

Ajouter et supprimer des éléments

- createElement()** - Crée un élément
- prepend()** - Ajoute l'élément entre parenthèses devant l'élément cible
- append()** - Ajouter l'élément entre parenthèses derrière l'élément cible (peut contenir du texte)
- appendChild()** - Ajouter l'élément entre parenthèses derrière l'élément cible (ne peut pas contenir du texte)
- insertBefore()** - Insère un élément avant l'élément cible

Modifier le style d'un élément

- style.*propriété*** - Modifie la propriété CSS spécifiée, par exemple : style.color = "orange"
- className** - Modifie les classes d'un élément

Pratique 06 : créer une page en javascript

Pour réaliser cet exercice, vous allez devoir :

- Supprimer la div avec l'id `#a-supprimer`
- Créer un header
- Créer un sous-header
- Créer un paragraphe

Tout ceci **dynamiquement** grâce à JavaScript.

Codes couleurs :

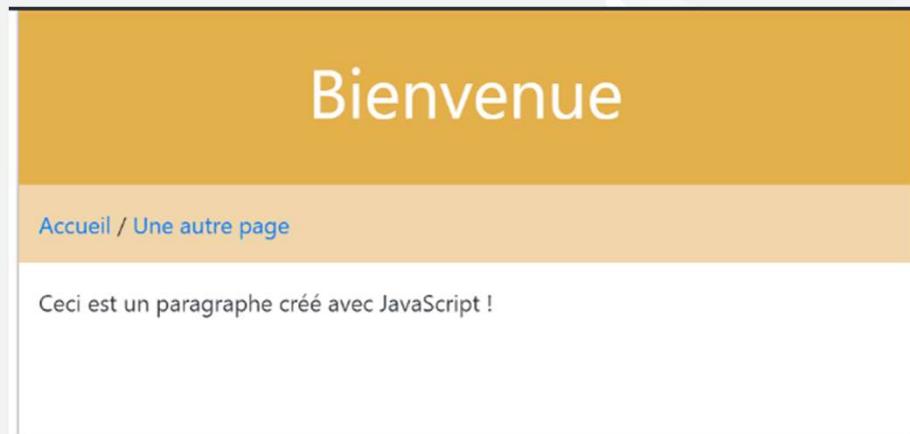
Je vous invite à vous lancer dans votre imagination, mais toutefois, je sais à quel point parfois, nous ne voulons pas nous prendre la tête ! Si vous le souhaitez, vous pouvez donc reprendre mes couleurs :

- header - `#e3b04b`
- sous-header - `#f1d6ab`

Je ne vous aide pas plus, j'ai juste trois règles pour réussir ce projet :

1. Votre fichier CSS doit rester vide
2. Votre fichier HTML ne doit pas être modifié (donc vous ne devez pas toucher au code déjà présent)
3. Amusez-vous ! Ne vous bloquez pas : "Est-ce que je pourrai faire plus simple ?", le principal, c'est que vous réussissiez ce projet :)

Bon courage !



Voir Pratique 06.js

Les événements

La plupart des programmes Javascript embarqués dans une page web sont guidés par des **événements**. Un **événement** est le moyen de faire réagir le programme Javascript à quelque chose qui se produit dans son environnement (l'utilisateur, le navigateur, la page html, le Javascript lui-même...), par exemple:

- l'utilisateur clique sur un lien, déplace la souris, entre une donnée dans un formulaire, frappe une touche au clavier...
- un document, une image ou une iframe vient d'être chargé dans le navigateur...
- une erreur s'est produite, un temps d'attente est dépassé...
- des données viennent d'être reçues en [AJAX](#)...

C'est en effet l'apparition de ces événements qui vont piloter le déroulement du programme, plutôt qu'une exécution linéaire comme dans la plupart des programmes classiques

Un **événement** est quelque chose qui se produit et qui demande qu'une action soit prise

- Un **gestionnaire d'événement** ou un **observateur d'événement** est la partie du code qui sera exécutée afin de réaliser l'action souhaitée

Les événements

L'action par défaut d'un événement

Certains événements sont liés à une **action par défaut**

Par exemple:

- si on clique sur un lien hypertexte, l'action par défaut sera de suivre le lien en question
- si on clique sur une checkbox, l'action par défaut sera de sélectionner ou de désélectionner la checkbox
- si on clique sur un bouton pour envoyer un formulaire, l'action par défaut sera d'envoyer le formulaire

Les gestionnaires et les observateurs d'événement seront en mesure d'annuler cette **action par défaut** (si le type d'événement le permet). Si elle n'est pas annulée, l'action par défaut sera exécutée à la toute fin du traitement de l'événement (après avoir exécuté les autres actions éventuelles définies dans les gestionnaires et les observateurs d'événement)

L'élément cible d'un événement

La plupart des événements vont avoir pour cible un des éléments de la page html (un lien hypertexte, un zone dans un formulaire, une div...). On parlera alors de l'**élément cible** de l'événement (*target element*)

A titre d'exemple, si on clique n'importe où dans la page avec le bouton gauche de la souris, un événement appelé **click** se produira. Cet événement aura pour élément cible l'élément html qui englobe au plus près l'information sur laquelle on a cliqué

Les événements

Les écouteurs (événement 'on')

Les écouteurs 'on' sont des attributs que je peut placer directement dans mon html pour y injecter du code js. C'est une technique qui tends à disparaître, plutôt que d'injecter du code js dans la balise html, on va plutôt utiliser les propriété js (vues précédemment) pour obtenir un résultat identique.

```
12 <a onclick="return confirm('Etes-vous sûr de vouloir supprimer cet article ?')" href="http://google.be">Supprimer cet article</a>
13
14 <button onmouseover="document.body.style.backgroundColor='orange'"
15   onmouseout="document.body.style.backgroundColor='white'">Passez au-dessus de moi</button>
16
```

Les écouteurs via les propriétés js

On va faire la même chose que précédemment mais en utilisant les propriétés js:

```
21 <a class='hypperlien' href="#">vers le site de l'ifapme</a>
22 <button class="bouton">Passez moi dessus (en tout bien tout honneur)</button>
23
```

```
1 let a = document.querySelector('.hypperlien');
2 let button = document.querySelector('.bouton');
3
4 a.onclick = () => {
5   if(confirm("Etes-vous certain?")){
6     location.href="https://www.ifapme.be/";
7   }
8 }
9
10 button.onmouseover = () => {
11   document.body.style.backgroundColor = "blue";
12 }
13
14 button.onmouseout = () => [
15   document.body.style.backgroundColor = "white";
16 ]
```

Voir exemple20.html et exemple20.js Partie 1

Les événements

Les écouteurs via le gestionnaire d'événements

La méthode pour moi la plus pratique, nous avons à notre disposition un gestionnaire d'événements qui va nous permettre d'ajouter ou de retirer un évènement à volonté grâce à la méthode **addEventListener()**. Du coup on repart dans notre exemple

```
22 let a2 = document.querySelector('.hyperlien2');
23 let button2 = document.querySelector('.bouton2');
24
25 a2.addEventListener('click', () => {
26   if(confirm('Êtes-vous certains?')){
27     location.href="https://www.ifapme.be";
28   }
29 })
30
31 button2.addEventListener('mouseover', ()=>{
32   document.body.style.backgroundColor = 'green';
33 });
34
35 button2.addEventListener('mouseout', () =>{
36   document.body.style.backgroundColor = 'white';
37 }
38 )
```

La méthode **addEventListener()** comprends deux paramètres:

- 1° l'évènement à écouter
- 2° la fonction à exécuter!

Voir exemple20.html et exemple20.js Partie 2

Les événements

Retirer un listener

On ne peut pas retirer un listener à proprement parler, mais on peut lui retirer une fonction qu'on lui a associée. Pour réaliser ça, je ne pourrai plus déclarer de fonction anonyme dans mon listener, mais bien coder une méthode que je pourrai appeler via le listener et que je pourrai enlever à loisir avec la méthode **removeListener('evenementAEcouter', fonction à enlever)**

```
40 // retirer un événement
41
42 let button3 = document.querySelector('.bouton3');
43
44 function changerBgAqua(){
45     document.body.style.backgroundColor = 'aquamarine';
46 }
47
48 function changerBgWhite(){
49     document.body.style.backgroundColor = 'white';
50 }
51
52
53 button3.addEventListener('mouseover', changerBgAqua); //J'appelle la fonction sans mettre les ()
54
55 button3.addEventListener('mouseout', changerBgWhite);
56
57 button3.removeEventListener("mouseout", changerBgWhite);
58 |
```

Voir exemple19.html et exemple19.js

setTimeout et setInterval

Ces deux méthodes vont nous permettre d'exécuter du code à un moment précis. Ils prennent tout deux deux paramètres: d'abord le code ou la fonction à exécuter, ensuite le délai en milliseconde ou le navigateur devra exécuter ce code.

```
1 // setTimeout  
2 let timer = setTimeout("alert('Bonjour')", 3000);  
3 clearTimeout(timer);  
4  
5 // setInterval  
6 let interval = setInterval("alert('Bonjour')", 5000);  
7 clearInterval(interval);|
```

ATTENTION: setTimer va exécuter le code une seule fois, setInterval lui va répéter ce code indéfiniment en prenant toujours en compte le délai qui lui a été programmé! Pour arrêter cet interval on utilisera la méthode clearInterval()

```
5 // setInterval  
6 let interval = setInterval("alert('Bonjour')", 5000);  
7 clearInterval(interval);  
8  
9 setTimeout(clearInterval(interval), 25000); //Au bout de 25s j'arrête ma fonction setInterval|
```

Voir exemple21.js

Exercice 04 : faire un timer

Dans cet exercice, vous allez réaliser un Timer !

Nous allons pouvoir réviser les notions de conditions, de fonctions, et d'intervalles.

Voici comment va se dérouler notre Timer :

- Un bouton s'affiche sur la page proposant de « **Lancer le décompte** »
- L'utilisateur clique sur ce bouton, qui va lancer une fonction **start()**, qui s'occupera de créer un intervalle sur la fonction **decompte()** toutes les secondes, pendant 10 secondes.
- La fonction **decompte()**, lorsqu'elle est appelée :
 - Décrémentera (retirera 1) à une variable **secondes**, qui contient comme valeur 10
 - Vérifiera si **secondes** est différent de 0, pour afficher dans la page le nombre de secondes restantes
 - Sinon, si **secondes** vaut 0, appellera la fonction **stop()**, qui aura pour but d'arrêter l'intervalle

[Voir exercice04.html et exercice04.js](#)

Exercice 04 : faire un timer

Etape 1

Créez un bouton, avec le texte « Lancer le décompte », dans votre page HTML.

Etape 2

Récupérez grâce à JavaScript, votre élément *button*, et stockez-le dans une variable *btn*.

Etape 3

Ajoutez un évènement sur votre bouton, qui se déclenchera au clic, et qui appellera la fonction *start()*.

Etape 4

Créez une fonction *start()* qui aura pour but de créer une variable stockant un intervalle. Cet intervalle devra appeler une fonction *decompte()* toutes les secondes. **Je vous conseille d'initialiser toutes vos variables à l'extérieur de vos fonctions pour pouvoir les utiliser partout (hors, évidemment, la variable qui contient notre intervalle).**

Etape 5

Créez une fonction *decompte()* qui décrémentera dans un premier temps la variable *secondes*. Créez cette variable en-dehors de votre fonction. Puis, vérifiez si *secondes* vaut 0. Si c'est le cas, appelez la fonction *stop()*, sinon, affichez dans la page HTML, grâce à JavaScript, le nombre de secondes restantes (par exemple, s'il reste 5 secondes, écrivez « 5 » dans la page, à la suite des autres secondes, comme dans l'exemple ci-dessus).

Etape 6

Créez une fonction *stop()* qui arrêtera l'intervalle et qui se chargera d'écrire « Stop ! » dans la page.

Exercice 04-B : faire horloge

Etape 1

Créez un bouton, avec le texte « Lancer le décompte », dans votre page HTML.

Etape 2

Récupérez grâce à JavaScript, votre élément *button*, et stockez-le dans une variable *btn*.

Etape 3

Ajoutez un évènement sur votre bouton, qui se déclenchera au clic, et qui appellera la fonction *start()*.

Etape 4

Créez une fonction *start()* qui aura pour but de créer une variable stockant un intervalle. Cet intervalle devra appeler une fonction *decompte()* toutes les secondes. **Je vous conseille d'initialiser toutes vos variables à l'extérieur de vos fonctions pour pouvoir les utiliser partout (hors, évidemment, la variable qui contient notre intervalle).**

Etape 5

Créez une fonction *decompte()* qui décrémentera dans un premier temps la variable *secondes*. Créez cette variable en-dehors de votre fonction. Puis, vérifiez si *secondes* vaut 0. Si c'est le cas, appelez la fonction *stop()*, sinon, affichez dans la page HTML, grâce à JavaScript, le nombre de secondes restantes (par exemple, s'il reste 5 secondes, écrivez « 5 » dans la page, à la suite des autres secondes, comme dans l'exemple ci-dessus).

Etape 6

Créez une fonction *stop()* qui arrêtera l'intervalle et qui se chargera d'écrire « Stop ! » dans la page.

Exercice 05 : Bouton spoiler

Dans cet exercice, vous allez réaliser un bouton Spoiler !

Nous allons pouvoir réviser les notions de conditions, et de styles.

Voici comment va se dérouler notre bouton Spoiler :

- Un bouton s'affiche sur la page proposant d'afficher le message
- L'utilisateur clique sur le bouton, ce qui va activer une condition
 - La variable *hidden* vaut *true* ? Dans ce cas, on affiche le message, on change le texte du bouton en « Cacher », et on passe la variable *hidden* en *false*—
 - La variable *hidden* vaut *false* ? Dans ce cas, on cache le message, on change le texte du bouton en « Afficher », et on passe la variable *hidden* en *true*

[Voir exercice05.html et exercice05.js](#)

Exercice 06 : générateur de citations

A partir d'un tableau js et du code html donné par votre dévoué instructeur, créez un générateur de citation qui générera une nouvelle citation à chaque fois que vous cliquerez sur le bouton prévu à cet effet!

*“Pour gagner votre vie,
apprenez à l’école. Pour
gagner une fortune, apprenez
par vous-même.”*

Brian Tracy



Nouvelle Citation

Voir [exercice06.html](#) et [exercice06.js](#)

Exercice 06 : générateur de citations

Etape 1: codez le css de manière à faire correspondre votre page web au design ci-dessous. Au besoin, codez du contenu fictif dans la page html en attendant de le générer en js

Etape 2: Dans cette étape, la première chose que vous devez faire est de récupérer les éléments nécessaires au fonctionnement de notre générateur de citations. Ici, vous devez donc récupérer :

- l'emplacement qui contient la citation
- l'emplacement qui contient le nom de l'auteur
- L'emplacement qui contient l'image de l'auteur
- le bouton qui permet de générer une nouvelle citation

Etape 3:Dans cette étape, vous allez devoir créer les variables qui vont seront utiles pour faire fonctionner votre projet. Je vous recommande de déclarer vos variables au début de votre fichier JavaScript, pour que vous puissiez les utiliser partout dans votre fichier ensuite. Pour ma part, j'ai déclaré deux variables :
•**dernier** - contient l'index de la question actuellement affichée (par défaut, la citation affichée est la première du tableau, sa valeur est donc égale à 0)
•**nombreAleatoire** - contient le nombre aléatoire généré

Etape 4: Il est temps maintenant de détecter lorsque le bouton "Nouvelle Citation" est cliqué. Pour cette étape, je ne vais pas vous aider plus, essayez de reprendre ce que nous avons vu dans les sessions précédentes avec les évènements si vous n'y arrivez pas.

Etape 6: Voici donc venu la dernière étape de notre projet ! Nous avons réussi à générer un nombre aléatoire non-utilisé par la précédente citation, il ne faut plus que :

- Stocker ce nombre dans la variable *nombreAleatoire*
- Modifier le contenu de notre objet *citation* par la citation à l'index *nombreAleatoire* de notre tableau
- Modifier l'auteur de notre objet *auteur* par l'auteur à l'index *nombreAleatoire* de notre tableau

Si vous avez du mal, essayez de revenir sur la session liée aux tableaux multi-dimensions !

Etape 5: Votre bouton vient d'être cliqué, et vous venez d'être alerté par JavaScript, parfait ! Maintenant, nous allons devoir générer un nombre aléatoire, qui nous permettra d'avoir un nombre entier entre 0 et 5, si nous avons 6 éléments dans notre tableau.

Je vous recommande d'utiliser une boucle *do...while* pour cette étape. Ceci vous permettra de générer un nombre tant que ce dernier est égal au nombre de la variable *dernier*. Nous ne voulons pas afficher deux fois de suite la même citation si ?

Pour générer un nombre aléatoire avec JavaScript, il faut utiliser la fonction *Math.random()*. Le problème de cette fonction, c'est qu'elle génère un nombre à virgule (donc un flottant), entre 0 et 1. Nous, nous souhaitons un entier, entre 0 et 5 si nous avons 6 citations. Dans ce cas, il va falloir changer de fonctionnement !

Pour cette raison, je vous ai concocté cette fonction *genererNombreEntier(max)*, qui prend en paramètre le **nombre d'éléments dans votre tableau de citations**. N'hésitez pas à la réutiliser ! Cette fonction utilise la fonction *Math.floor()* qui renvoie le plus grand entier qui est inférieur ou égal à un nombre. Par exemple, si je fais *Math.floor(5.8)*, elle me renvoie 5.

Ici, nous générerons donc un nombre aléatoire, par exemple 0.7, que nous multiplierons par notre nombre d'éléments. Si j'ai 6 éléments, j'ai donc $0.7 * 6$, ce qui fait 4.2. Ma fonction *Math.floor()* me retournera donc 4. Parfait : nous avons bien notre entier.