



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
17 </body>
18 </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>
```

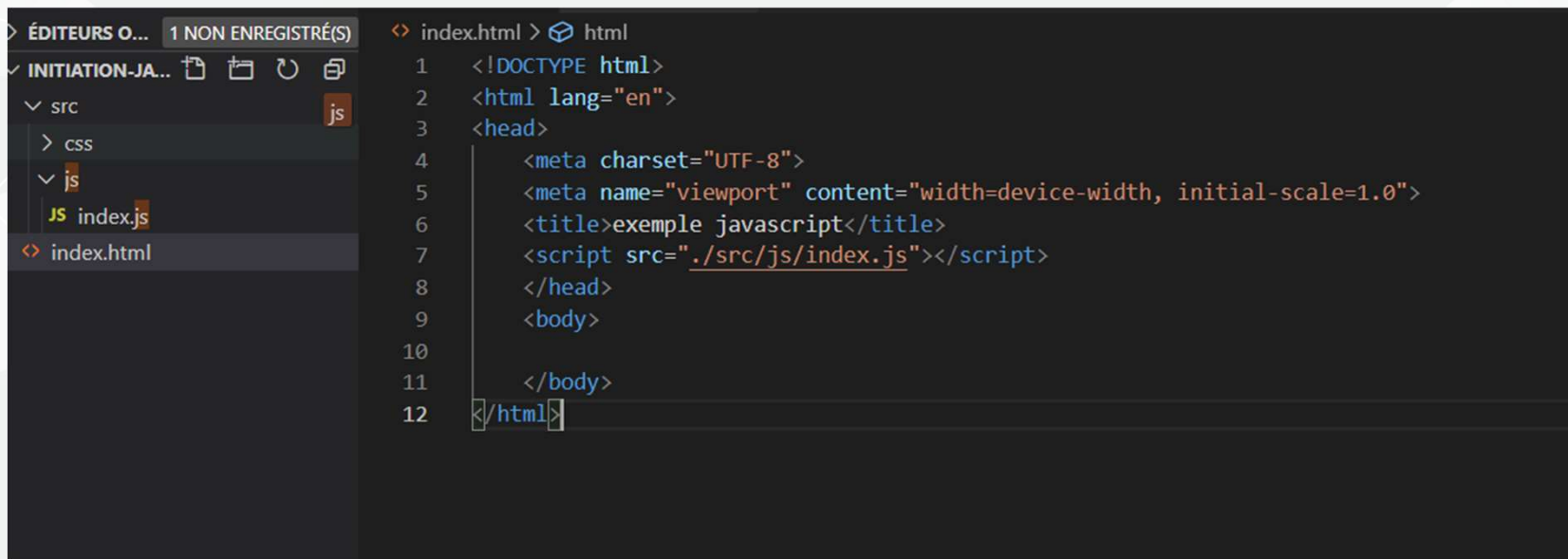
L'attribut **language** assurait la compatibilité dans les vieux navigateurs. Il n'est plus supporté actuellement:

```
<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



```
> ÉDITEURS O... 1 NON ENREGISTRÉ(S)
✓ INITIATION-JA...
  > src
    > css
    > js
      JS index.js
  <> index.html

<> index.html > html
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
11 </body>
12 </html>
```

On place 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)

On place 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**, **onkeypressed**, **onchange**...

Dans cet exemple, le gestionnaire d'évènement **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 retournée.

```
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 « **getElementsByTagName** » renvoi 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 apparaitre dans une boite de dialogue, grâce à la fonction `alert(...)`. Si l'anglais est sélectionné, c'est le message "Hello [votre prénom] [votre nom] !" qui doit apparaitre.

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

<code>window</code>	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...)
<code>document</code>	représente la page chargée dans le navigateur, c'est un objet de type HtmlDocument qui dérive du type Document de DOM
<code>JSON</code>	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 normalisée 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 HtmlDocument qui dérive de Document)
<code>.screen</code>	renseigne sur l'écran de l'utilisateur (taille...)
<code>.location</code>	renseigne sur l'adresse de la page chargée dans la 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 sauvegarder/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 window qui la représentait existe toujours)
<code>.frames</code> <code>.length</code>	un tableau reprenant les <frame> éventuelles, et le nombre de ces frames
<code>.top</code>	l'objet window de la fenêtre de plus haut niveau dans le cas, par exemple, où le document est découpé en frames (chaque frame aura son propre objet window)
<code>.opener</code>	l'objet window 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 window 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 découpé en frames (chaque frame aura son propre objet window)
<code>.self</code>	l'objet window de la fenêtre courante
<code>.window</code>	l'objet window lui-même
<code>...</code>	

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



<code>.alert(...)</code>	affiche une boîte de dialogue pop-up pour afficher un message à l'écran
<code>.confirm(...)</code>	affiche une boîte de dialogue pop-up afin de demander une confirmation à l'utilisateur
<code>.prompt(...)</code>	affiche une boîte de dialogue pop-up afin de demander à l'utilisateur d'entrer une valeur
<code>.open(...)</code>	ouvre une nouvelle fenêtre ou un nouvel onglet
<code>.close()</code>	ferme la fenêtre courante
<code>.focus()</code>	donne le focus à la fenêtre courante
<code>.blur()</code>	enlève le focus à la fenêtre courante
<code>.print()</code>	imprime le contenu de la fenêtre
<code>.setTimeout(...)</code>	lance l'exécution de code Javascript de manière différée dans le temps
<code>.clearTimeout(...)</code>	supprime une exécution différée demandée par <code>.setTimeout(...)</code>
<code>.setInterval(...)</code>	lance l'exécution de code Javascript à intervalles réguliers
<code>.clearInterval(...)</code>	supprime une exécution à intervalles réguliers demandée par <code>.setInterval(...)</code>
<code>.moveTo(...)</code>	déplace la fenêtre courante à une position donnée
<code>.moveBy(...)</code>	déplace la fenêtre courante d'une distance donnée
<code>.resizeTo(...)</code>	redimensionne la fenêtre courante à une taille donnée
<code>.resizeBy(...)</code>	redimensionne la fenêtre courante d'une distance donnée
<code>.scrollTo(...)</code>	scrolle la fenêtre courante à une position donnée
<code>.scrollBy(...)</code>	scrolle la fenêtre courante d'une distance donnée
<code>.postMessage(...)</code>	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>
18 </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)
- *défaul* 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 une interface standardisée utilisée 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:

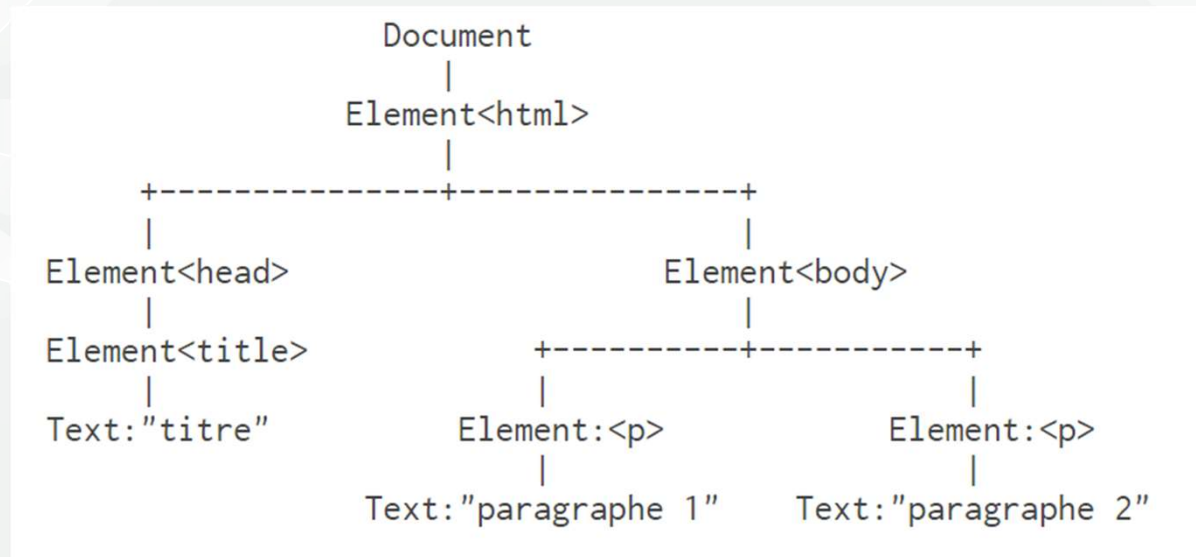
Document	pour modéliser le document lui-même
Element	pour modéliser un élément (une balise)
Attr	pour modéliser un attribut
Text	pour modéliser le texte contenu dans un élément (une balise)
...	

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 neud (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	le valeur associée au noeud (valeur de l'attribut, valeur du noeud texte, <i>null</i> 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(<i>new</i>)	ajoute un noeud <i>new</i> à la fin de la liste des enfants (retourne le nouveau noeud ajouté)
.insertBefore(<i>new</i> , <i>ref</i>)	ajoute un noeud <i>new</i> dans la liste des enfants, juste avant le noeud <i>ref</i> (retourne le nouveau noeud ajouté)
.removeChild(<i>old</i>)	efface le noeud <i>old</i> (retourne le noeud effacé)
.replaceChild(<i>new</i> , <i>old</i>)	remplace le noeud enfant <i>old</i> par le nouveau noeud <i>new</i> (retourne le noeud effacé)
.cloneNode(<i>deep</i>)	crée un clone du noeud courant (ainsi que tous ses descendants si <i>deep</i> 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:

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

Les Méthodes les plus importantes de Element

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

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

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 <code>undefined</code>

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}`);
11
```

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 charriot)
- > \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>	incrémement avant
<i>variable++</i>	incrémement après
<i>--variable</i>	décrémement avant
<i>variable--</i>	décrémement 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 mêler 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 [exmple12.js](#) partie 1

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

Déclare ma fonction

Nom de ma fonction

1^{er} et 2^{ème} paramètres de ma fonction

```
function sayHello(nom, prénom){  
  console.log(`Bonjour ${prénom} ${nom}`);  
}
```

Appel de ma fonction

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

exemple12-2.js partie 2bis

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  }
```

La fonction ci-dessus, nous sert à afficher le résultat d'une opération dans une boîte 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      console.log('je suis une fonction anonyme');  
3  }  
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

Les fonctions anonymes

Fonction anonyme dans une variable:

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

Pour l'utiliser

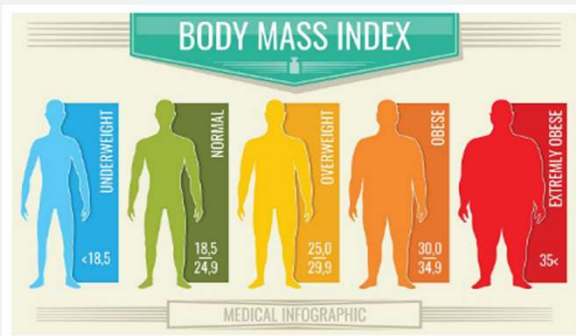
```
10 direBonjour();
```

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")}());
```



EXERCICE 01 – Calcul de l'ime

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'url photo 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 !