</code>mob

Coder

CodeMob: Codeur. Octobre 2017. http://codemob.eu/. Auteurs:

This publication has been co-funded by the European Commission's Ersamus+ Programme.

Table des matières

Bienvenue dans cette initiation à la programmation!

À la fin de ce cours, vous devriez être capable de :

- Créer un jeu (par exemple un memory) en JavaScript
- Ajouter une interface graphique (GUI: graphical user interface) avec HTML5 & CSS3
- Comprendre et expliquer les concepts de base de la programmation

Tout d'abord, gardez à l'esprit qu'il existe des milliers de ressources en ligne. Ce ne serait pas possible d'offrir un cours complet ici. Quoi qu'il en soit, dans la programmation, vous trouverez vos réponses via une simple recherche sur le web. Considérez ce cours comme une introduction, pas à pas, à JavaScript ! Profitez !

Pourquoi apprendre JavaScript?

À ne pas confondre avec Java, JavaScript vous permet de créer des sites Web interactifs. JavaScript est devenu une technologie Web essentielle avec HTML et CSS, car la plupart des navigateurs implémentent JavaScript. Ainsi, vous devez apprendre JavaScript si vous souhaitez accéder au développement web, et vous devez bien l'apprendre si vous envisagez de devenir un développeur front-end ou d'utiliser JavaScript pour le développement de back-end.

En outre, l'utilisation de JavaScript est maintenant étendue au développement d'applications mobiles, d'applications de bureau et au développement de jeux. Dans l'ensemble, il a explosé en popularité et est maintenant une compétence très recherchée.


3
Beginner Friendliness
3
Scalability
5
Community
5
Career Opportunities
5
Future

Source:

Algorithmique

L'algorithmique

"un algorithme est une suite finie et non-ambiguë d'instructions permettant de donner la réponse à un problème" - Wikipedia FR

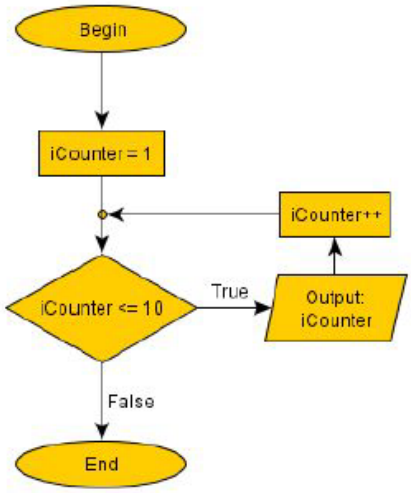
Deux manières courantes de le représenter : le pseudo-code et les organigrammes.

L'avantage d'apprendre les bases de l'algorithmique est de séparer la logique d'un quelconque langage ou de sa syntaxe.

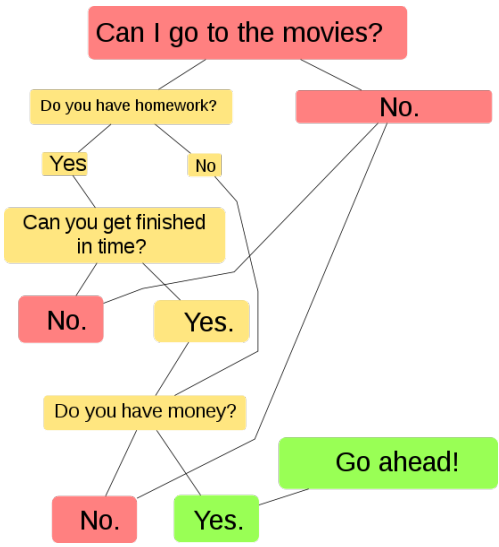
Notes : Pseudo-random

Sans y penser, nous en faisons souvent d'une certaine manière. Par exemple, une recette, une addition multiple faîte mentalement ou renseigner son chemin à un passant.

Démo : Expliquer un chemin (plusieurs solutions, notions de fonction et de refactorisation)



Voici un simple exemple de qu'un algorithme peut être :



Quelle est la relation entre un **algorithme** et un **programme**? Considérez un **algorithme** comme une formule pour résoudre quelque chose. Un **programme** est une série d'instructions pour l'ordinateur qui utilise des algorithmes pour exécuter la tâche souhaitée.

Comprenons ceci à travers un exemple:

Algorithme simple

Aire du rectangle A = longueur X largeur
Le programme de travail serait :

- Demandez à l'utilisateur de saisir la longueur
- Demandez à l'utilisateur de saisir la largeur
- Utiliser l'algorithme de l'aire pour calculer la superficie
- Afficher la réponse

Introduction aux langages HTML et CSS

Pour créer un site Web, vous devez fournir des informations à l'ordinateur. Il ne suffit pas de taper le texte qui sera dans son site, il est également nécessaire de savoir comment placer ce texte, insérer des images, créer des liens, etc. Pour expliquer à l'ordinateur ce que vous voulez, ce sera nécessaire d'utiliser un langage qu'il comprend.

Il existe des langages pour créer des programmes, tels que C ++ ou Java. Ces langages sont pourtant complexes et destinées aux personnes qui possèdent déjà des connaissances informatiques.

HTML et **CSS** sont précisément utilisés pour créer des sites Web, et ils ont été créés pour être simples à utiliser. Chacun de ces 2 langages est utilisé pour faire quelque chose de spécifique, et les deux se complètent naturellement pour finalement donner un site web:

• **HTML**: C'est l'abréviation de **HyperText Markup Language**. Il est apparu en 1991, lorsque le Web a été lancé. Son rôle est de gérer et d'organiser le contenu. C'est donc en HTML que vous écrivez ce qui doit être affiché sur la page: texte, liens, images… Vous direz par exemple: « Ceci est mon titre, c'est mon menu, voici le texte principal de la page... ».
Dans ce cours, nous travaillerons sur la dernière version de HTML (HTML5), qui est aujourd'hui le langage de l'avenir, que tout le monde est encouragé à utiliser.

• **CSS**: c'est l'abréviation de **CASCADING STYLE SHEETS**. Ce langage ne sert qu'à présenter la page Web. C'est dans le CSS qu'il sera dit: "Mes titres sont en rouge et sont soulignés, mon texte est dans la police ARIAL, mon nom est centré, mon menu a un fond blanc ..." etc. Avec ce langage, nous pourrons créer rapidement et simplement la mise en page de notre site.

L'éditeur

Une question que vous devriez réellement vous poser est: "De quel logiciel vais-je avoir besoin pour créer mon site?"

In fine même un programme très basique comme Notepad serait suffisant pour créer un site web! Mais pour nous faciliter la tâche, nous allons utiliser des programmes appelés éditeurs texte (Text editor) comme Notepad ++ (un premier avantage est la colorisation syntaxique qui augmente la lisibilité et permet de repérer plus facilement des erreurs).

Les navigateurs

Qu'est-ce qu'un navigateur?

Le navigateur web (Browser) est le programme qui permet de naviguer sur internet et de visionner des pages web. Le travail du navigateur est de lire les codes HTML et CSS que vous avez écrit et d'afficher ce qu'il représente. Si votre code CSS dit "Les titres doivent être rouge," alors le navigateur les affichera en rouge.

Parmi les nombreux navigateurs, voici les plus connus:
Internet Explorer - Mozilla Firefox - Opera - Netscape - Konqueror (Linux) - Lynx (Linux) - Apple Safari (Mac) - etc.

Créez votre premier document HTML

Qu'est-ce que l'HTML?

Comme indiqué ci-dessus, HTML est un langage de balisage pour décrire les documents Web (pages Web).

- HTML signifie Hyper Text Markup Language
- Un langage de balisage est un ensemble de balises
- Les documents HTML sont décrits par les balises HTML
- Chaque balise HTML décrit différents contenus de document

Un petit document HTML

Exemple

```
<!DOCTYPE html>
<html>
```


```
<head>
<title>Le titre dans l'onglet</title>
</head>
<body>

<h1>Mon premier titre</h1>
<p>Mon premier paragraphe.</p>

</body>
</html>
```

Explication de l'exemple

- La balise <!DOCTYPE html> définit ce document comme étant de l'HTML5 au navigateur
- Le texte entre les balises <html> et </html> décrit un document HTML
- Le texte entre les balises <head> et </head> offre des informations à propos du document
- Le texte entre les balises <title> et </title> donne le titre du document
- Le texte entre les balises <body> et </body> affiche le contenu visible de la page
- Le texte entre les balises <h1> et </h1> affiche le titre principal du document
- Le texte entre les balises et affiche un paragraphe.

Grâce à ces descriptions, un navigateur web pourra afficher correctement un document avec un titre et un paragraphe de texte.

Pour plus d'information regardez le document sur .

Les éléments HTML indispensables

Dans cette leçon, votre tâche sera de créer un simple document HTML qui contiendra le contenu suivant:

-

N'oubliez pas d'utiliser des commentaires

Un **commentaire** est une balise HTML (**tag)** d'une format légèrement différent des autres :

<!-- Ceci est un commentaire -->

Vous pouvez le mettre où vous voulez dans votre code source: il n'a pas d'impact visuel sur votre page mais vous pouvez utiliser des commentaires pour expliquer votre code, ce qui peut être utile lorsque vous ou quelqu'un d'autre éditera le code source dans le futur. Cela devient bien sûr de plus en plus utile au plus votre code grandit. Attention, tout le monde peut voir le code HTML d'une page web disponible en ligne simplement en cliquant sur le bouton droit et sélectionnant "Montrer le code source » (Show the source code of the page). Donc, n'y mettez jamais de données sensibles... et prenez soin de votre travail.

Créer un fichier CSS et l'utiliser pour styliser votre document HTML

Qu'est-ce que CSS?

- L'acronyme CSS signifie Cascading Style Sheets ou feuilles de style en cascade.
- Le CSS décrit comment les éléments HTML doivent être affichés à l'écran, sur papier (à l'impression) ou dans d'autres médias.
- Le CSS fait gagner beaucoup de travail en contrôlant le layout de plusieurs pages web en une fois.
- Les feuilles de style externes sont stockées dans des fichiers CSS (.css).

Pourquoi utiliser CSS?

Le CSS est utile pour définir le style et le visuel des pages web, ceci incluant le design, le layout et toutes leurs variations pour différents matériels (PC, smartphone) et tailles d'écran.

Le CSS résout un grand problème

L'HTML n'as JAMAIS été prévu pour formater le visuel d'une page web!
Il a été créé pour décrire sémantiquement son contenu, comme par exemple :

<h1>Ceci est un titre</h1>
Ceci est un paragraphe

Lorsque que des balises telles que <font> et des attributs tels que ‘color’ ont été ajoutés aux spécifications d’HTML3.2, cela a été le début d’un cauchemar pour les web developers. Le développement de grand sites web, où il fallait ajouter ces informations de nombreuses fois et ce dans chaque page, devenait un processus long et fastidieux avec de nombreux risques d’erreurs.

Pour résoudre ce problème, le World Wide Web Consortium (W3C) a ensuite créé le CSS pour définitivement enlever des page web toute information sur le style et le visuel.

CSS vous épargne beaucoup de travail!
Les définitions de style sont normalement sauvées dans des fichiers .css externes. Il est donc possible de changer le look d’un site web tout entier juste en modifiant un seul fichier!
Pour plus d’informations, rendez-vous sur .

Feuille de style externe (External style sheet)
Lorsqu’un navigateur lit une feuille de style, il va formater et mettre en page l’HTML en se basant sur les déclarations CSS qu’il y trouve.

Il y a 3 manières d’insérer du CSS :

1. Via une feuille de style externe

2. Via une feuille de style interne

3. Via du style inline (directement dans l’HTML)

Dans cette leçon, nous utiliserons la manière le plus conseillée, c.-à-d. une feuille de style externe.
Chaque page HTML doit inclure une référence vers le fichier de la feuille de style externe via la balise <link> qui doit se trouver dans le header (entre <head> et </head>) :

Exemple:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

Une feuille de style externe peut, elle aussi, être écrite dans n’importe quel éditeur de texte. Elle n’a pas à contenir d’HTML, uniquement du CSS et doit être sauvée sous l’extension .css.

Voici à quoi ressemble le contenu du fichier "myStyle.css" :

```
body {
    background-color: lightblue;
}
h1 {
    color: navy;
    margin-left: 20px;
}
```

Note: n'ajoutez pas d'espace entre la valeur de la propriété et l'unité (telle que margin-left: 20 px;). La manière correcte est : margin-left: 20px;

Programmer en JS

Histoire de JavaScript

Le langage JavaScript a été créé en **1995** par **Brendan Eich** (Fondation Mozilla) pour le compte de Netscape. Le langage, actuellement à la version 1.8.5, est une implémentation de la 3e version de la norme ECMA262.

- En décembre 1995, Sun et Netscape annoncent la sortie de JavaScript. Microsoft réagit alors en développant JScript (Même langage pour un nom différent pour éviter une dispute de Trademark avec Sun).
- Netscape soumet JavaScript à Ecma International pour standardisation en novembre 1996.
- **DHTML** (1996) ; un début difficile et inutile
- **Ajax** (2000) ; Un regain d'intérêt grâce à de nouvelles possibilités
- **JSon** ; une alternative à XML basée sur JavaScript
- **Frameworks JavaScript** ; Facilité, productivité et standardisation mais choix difficile
- **node.js** ; Eventdriven & Serversided
    - NodeWebkit & Cordova

Qu'est-ce que JavaScript?

- Langage de script (langage de haut niveau)
    - Interprété (en opposition aux langages compilés)
- Langage côté client (Il s'exécute sur la machine de l'utilisateur et non sur le serveur)
    - Néanmoins node.js apparu en 2009 change quelque peu la donne.
- Capable de changer le DOM :
    - Créer, modifier et supprimer les éléments HTML, leurs attributs ou le CSS.

- Créer, écouter et supprimer les événements.
- Rien à voir avec le langage Java de Sun... ;)

Exemple : les Polyfills &

Installation

Les scripts peuvent être placés dans une page HTML soit dans le <head>, soit dans le
<body> (juste avant le </body>) et Ils peuvent être **internes ou externes**.
L'attribut **type est optionnel** car sa valeur par défaut est la bonne (JavaScript est le seul langage accepté en HTML).

```
<script type="text/javascript">
//...(Votre code JS)
</script>

<!-- S'il y a un source externe la balise doit être vide -->
<script src="./js/script.js"></script>
```

Exemple JavaScript

Essayons de faire notre premier script JavaScript. Dans cet exemple, nous allons créer une simple fenêtre d'alerte (alert box). Nous allons accomplir cela en liant un fichier externe .js à notre document HTML.

La première chose à faire est d'aller dans le répertoire où vous avez déjà créé vos fichier HTML et CSS. Créez-y un nouveau fichier.js et sauvez-le au même endroit sous (par exemple) *myscript.js*.

Copiez-y le script suivant :

```
alert("Je suis un message d'alerte!");
```

Liez myscript.js à votre document HTML :

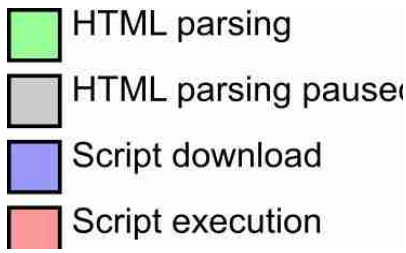
```
<!DOCTYPE html>
<html>
<body>
<script src="myScript.js"></script>
</body>
</html>
```

il est possible de placer la référence vers un fichier JS dans le <head> ou le <body> comme vous voulez. Le comportement sera identique.

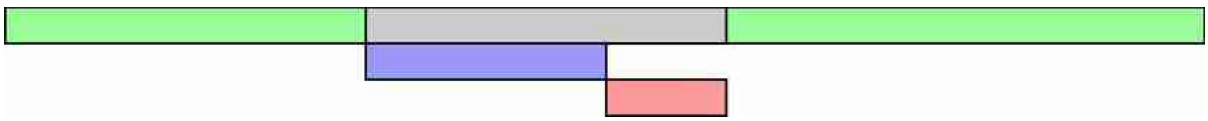
Ordre d'exécution des scripts - Defer & async

Le moment de l'exécution d'un script dépend de sa position dans la page HTML et des attributs defer et async.

Les attributs defer et async sont uniquement pris en compte pour les scripts **externes**

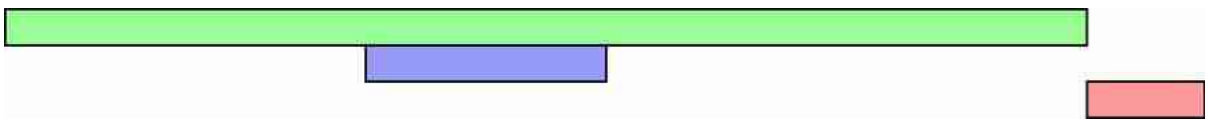


Exécution normale



L'utilisation courante : si deux scripts se suivent, le deuxième ne s'exécutera jamais avant la fin du premier car toute ressource est 'bloquante'.

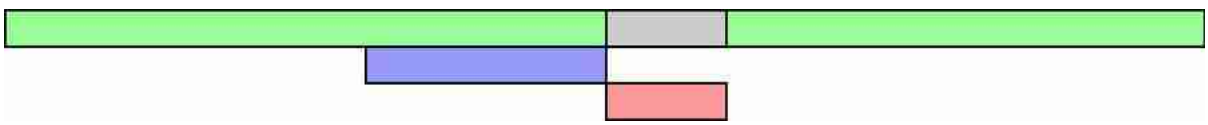
Defer



N'utilisez **jamais** de **document.write()** ou équivalent (avec defer).

Faîtes toujours attention à l'ordre d'exécution des scripts ayant la même priorité ; il est censé être respecté mais bugge dans IE4-9.

Async (HTML 5)



Idéal pour un script dont le moment d'exécution n'est pas important (par ex. : Google Annalytics) mais l'ordre d'exécution des scripts n'est pas garanti!

N'utilisez **jamais** de **document.write()** ou équivalent (avec async).

Layout Trashing

Lorsque JavaScript lit et réécrit ou change le DOM, le layout est invalidé et doit être recalculé à un certain moment dans le futur. Les navigateurs essaient d'attendre la fin d'une frame mais s'ils doivent le faire avant, cela risque d'avoir un impact sérieux sur les performances.

Bref, utilisez le moins possible les manipulations et changement du DOM, ou au pire, regroupez-les le plus possible.

Si le sujet vous intéresse, vous pouvez vous renseigner sur requestAnimationFrame et des librairies comme FastDOM (https://github.com/wilsonpage/fastdom).

La console JavaScript

Il n'est pas facile de savoir ce qui ne fonctionne pas dans un projet JavaScript sans un debugger.
Le Debugging est le processus de tester, trouver et supprimer les erreurs (bugs) dans un programme informatique.

Si un script ne fonctionne pas (bug), on ne voit que rarement l'erreur... Il faut donc afficher la console Javascript pour voir les **messages d'erreurs** (fichier & ligne de code, code couleur, script à la volée).

- **Firefox** :F12 > Console (JavaScript)
- **Chrome** :Ctrl+Shift + I > Console

On peut aussi débugger avec **console.log(...);** qui accepte en paramètre ce qui doit être affiché. Il est aussi possible de rajouter des **break points** ou d'utiliser la commande **debugger;**.

La méthode console.log()

Si votre navigateur supporte des options de debugging, vous pouvez utiliser la méthode console.log() pour afficher des valeurs JavaScript dans la console :

Exemple

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>

<script>
a = 5;
b = 6;
c = a + b;
console.log(c);
</script>

</body>
</html>
```

Paramétrer les Breakpoints

Dans la fenêtre de debug, il est possible de paramétrer des points d'arrêt (breakpoints) dans le code JavaScript.

A chaque point d'arrêt, l'interpréteur JavaScript va se mettre en pause, et vous laisser l'occasion d'examiner les valeurs actuelles de certaines variables.

Après cela, vous pouvez continuer l'exécution du code (généralement via un bouton Play).

Le mot-clé debugger

Le mot-clé **debugger** stoppe l'exécution de JavaScript et appelle les fonctions du débuggeur.

Cela a exactement le même effet que de paramétrer un point d'arrêt manuellement dans la console de debug.

Sans console ou outil de debug, ce mot-clé n'a aucun effet.

Dans l'exemple ci-dessous, ce code va se mettre en pause avant d'exécuter la troisième ligne de code.

```
var x = 15 * 5;
debugger;
document.getElementbyId("demo").innerHTML = x;
```

Gardez à l'esprit que les fonctionnalités de la console ne font pas partie des standard du langage même s'ils sont implémentés dans, par exemple, Firefox et Chrome.

Exemple API Console Firefox

Les commentaires

Les commentaires servent **à expliquer le sens du code JavaScript**, et à le rendre plus lisible.

Les commentaires peuvent aussi servir **à temporairement désactiver un bout de code**, lors d'un test ou d'une modification plus importante.

Les commentaires sur une seule ligne

Les commentaires sur une seule ligne commencent par //.

Tout texte après // sera ignoré par JavaScript (ne sera pas exécuté).

Cet exemple utilise un commentaire sur une seule ligne avant chaque ligne de code :


```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Cet exemple utilise un commentaire sur une seule ligne à la fin de chaque ligne de code à expliquer :

```
var x = 5;       // Declare x, give it the value of 5
var y = x + 2;   // Declare y, give it the value of x + 2
```

Commentaires multi-lignes

Les commentaires multi-lignes commencent par /* et finissent avec */.

Tout texte entre /* et */ sera ignoré par JavaScript (ne sera pas exécuté).

Cet exemple utilise un commentaire multi-lignes pour expliquer le code qui suit :

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Utiliser des commentaire pour empêcher l'exécution d'un bloc de code

Utiliser des commentaires est souvent utile pour tester et activer ou désactiver rapidement un bout de code.

Il suffit d'ajouter par exemple // devant une ligne de code pour la convertir en commentaire et pouvoir la conserver pour une utilisation future.

Cet exemple utilise les // pour empêcher l'exécution d'une ligne de code :

Example

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Cet exemple utilise un bloc de commentaire pour empêcher l'exécution de plusieurs lignes de code :

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

Les commentaires sur une seule ligne commencent par **//** et ceux sur plusieurs lignes sont entourés par **/***...***/**. Ci-dessous, un cas légèrement plus compliqué pour insérer de manière valide du JavaScript dans un document XHTML.

Utilisez **toujours des commentaires** pour commenter **votre logique**, les paramètres attendus par une fonction, ce qu'elle retourne ou désactiver un bout de code sans l'effacer.

```
<scripttype="text/javascript">
//<![CDATA[
varmonPremierMessage ='Hello World';
```


```
console.log(monPremierMessage);

functionmaPremiereFonction(){
    vardemo =document.getElementById("demo"); demo
    .style.color ="#990000";
}

maPremiereFonction();
//]]>
</script>
```

Les choses à ne pas faire: Eval & javascript:

N'utilisez (quasi) jamais **eval()**. Comme il permet d'exécuter un code arbitraire, il représente un risque de sécurité.

La fonction **Setinterval()** qui appelle son premier paramètre après un certain laps de temps utilise eval() si elle reçoit une chaîne de caractères en paramètre. Préférez donc une fonction comme argument.

```
setInterval( function(){    myFunction(param1,param2);},5000);
```

Conservez aussi bien la séparation entre HTML et JavaScript (comme avec le CSS pour des raison de maintenance et de logique) en évitant les document.write(...) et en séparant le JavaScript du layout.

```
<! Ne faîtes jamaisceci... > <ahref=
"javascript:myFunction();">....</a>

<! ...et évitez le plus possible ceci > <atitle="Click to
do some JS stuff" href="enablejs.html" onclick=
"MyFunction();return false;">link</a>
```

Les opérateurs mathématiques

- *, /, +, (ordre de précédence & parenthèses)
- % (Modulo)
- +=, =, *= & /= (x = x + 5;)
    - x++ & x (postincrément ou décrément : x = x + 1)
    - ++x & x (preincrément ou décrément)

Les constantes

Similaires aux variables à l'exception du fait qu'une fois déclarées leur valeur ne peut plus être changée. Elles sont souvent utilisées comme paramètres prédéfinis.

Par convention et dans quasi tous les langages, les noms de constantes sont écrits en majuscules séparés par un blanc souligné (underscore). Attention, il n'y pas de support des constantes sur IE10

```
// Constante
constMY_FAVORITE_NUMBER =9;

// 'Fausse' constante par convention uniquement
var MY_LUCKY_NUMBER =7;
```

Les variables & leur data type

JavaScript est un langage à **typage dynamique** (Dynamic typing) comme le PHP, utilise une **syntaxe pointée** et est **case sensitive** ; soyez donc méthodique et rigoureux car le type d'une variable risque de changer au run-time.

On considère généralement comme 'best practice' l'utilisation du Lower Camel Case pour les noms de variables (première lettre de chaque mot sauf le premier en majuscule, pas d'espace).

JavaScript possède un ramasse-miettes (Garbage Collector) qui détruit les variables qui ne sont plus utilisées (ou mise à null).

N'utilisez pas les syntaxes de type var a = new Array(); ni de tiret dans les noms de variables (par exemple : var cause-Une-Erreur = 5;).

```
varuserIdNumber; // déclaration de variable avec la balise var
userIdNumber =5; // Initialisation de la même variable

varuserIdNumber =5; // déclaration and initialisation simultanées

varvariable1 ="Hello World!";
varvariable2 =42;


varvariable1 ="Hello World!",
    variable2 =42;          // Déclaration multiple
```

Data type

- Primitive
  - **Strings** (Chaînes de caractères) : "Hello World"
  - **Numbers** (Nombres entiers et à virgule flottante) : 3.14 ou 42
  - **Booleans** (Valeurs booléennes) : true ou false
  - undefined (Une variable qui n'est pas encore déclarée ou ne correspond à aucun type)

    - null (Variable vide, càd à laquelle on a assigné une valeur nulle)
- **Objects** (Objets)
    - **Functions**(Fonction)
    - **Arrays** (Tableaux)
    - Date, Math
    - RegExp (Expressions régulières)

Casting

Il est possible de changer le type d'une variable (Casting) vers un autre via **Number**(...), **String**(...) et **Boolean**(...) ou même certaines méthodes comme .split() ou .join().

```
Var nbr2Boo = Boolean(0);          // false
var str2Nbr =Number('12');         // 12
var boo2Str =String(true);         // "true"
```

Strings

- La concaténation de chaînes s'exécute via le signe « **+** »
- Les chaînes de caractères sont délimitées soit par un **double guillemets "** (double quote) ou un simple guillemets **'** (single quote).
    - Au contraire du PHP, il n'y a pas de différence entre les deux.
    - Les caractères spéciaux peuvent être insérés via **** (Escape character)
        - \", \n (retour à la ligne), \\, ...
        - \ à chaque retour à la ligne
- La longueur est accessible via la propriété **.length**
- Les chaînes possèdent de nombreuses méthodes ;
    - .charAt(); indexOf(), substr, substring(), toLowerCase(), toUpperCase(), ...
- N'accédez pas à une chaîne de caractères comme si c'était un Array.

Exemple

Numbers

- Un seul type (Integer & Float) stocké sous un 64bit Floating Point.
- Les constantes Infinity & -Infinity représentent évidemment l'infini positif et négatif.
- **NaN** (Not a Number) sera renvoyé pour les impossibilités mathématiques comme une division par une chaîne de caractères par exemple.
- La méthode **.toString()** les convertit en chaînes de caractères
    - .toString(2) : conversion en binaire

  - .toString(16) : conversion en hexadécimal
- L'objet **Math** offre de nombreuses fonctionnalités comme :
  - Math.random() : Retourne un chiffre aléatoire entre 0 inclusif et 1 exclusif
  - Math.min(..., ...) & Math.max(..., ...) : Minimum & Maximum
  - Math.round(), Math.ceil() & Math.floor() : arrondi, arrondi vers le haut et vers le bas
  - Main.sin(), Math.cos(), Math.PI, ...

Exemple http://www.w3schools.com/js/js_number_methods.asp

Exercice Affichez aléatoirement le chiffre 0 ou 1 dans la console à chaque reload de la page (F5).

Array

- Les tableaux sont instanciés via des crochets vides **[ ]**(Brackets) et les utilisent pour accéder à un index bien précis.

  - ```
    varmyArray =[];// Nouveau tableau vide
    ```

- L'index commence comme dans quasiment tous les langages informatiques à 0.

  - ```
    myArray[0]=42;
    ```

- Il est possible de trouver la longueur d'un tableau via la propriété **.length**
  - Ne modifiez jamais un Array dans lequel vous itérez.
- Attention, ce ne sont pas des tableaux associatifs (Associative Arrays)
- Array possède des méthodes pour à peu près tout : .push(), .pop(), .splice(...), ...
  - Il est possible de trier les chaînes de caractères d'un tableau via la méthode **.sort()**
  - et les chiffres via .sort() utilisant une fonction de comparaison en argument **.sort(function(a, b) {return a b})**

Exemples: W3Schools & www.w3schools.com/js/js_array_methods.asp

Exercice Affichez le dernier élément d'un array dont vous ne connaissez pas la longueur.

Tester un type de variable

Il existe de nombreuses méthodes pour tester le type d'une variable.

- **typeof** maVariable
    - `typeofmyVar ==='undefined'`
    - Le data type de NaN est 'number'
    - Le data type d'un tableau (Array) est 'object'
    - Le data type de l'objet Date est 'object'
    - Le data type de null est 'object'
    - Le data type d'une variable indéfinie est 'undefined'

Exemple

-

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object" (not "array", see note
below)
typeof {name:'John', age:34} // Returns "object"
```

The typeof operator returns "object" for arrays because in JavaScript arrays are objects.

isNaN(…)

- isFinite(…)
- Array.isArray(…);
- …

Var ou pas var?

Dans une fonction, une **variable** déclarée **avec var est locale** et globale si elle est déclarée sans. Pour faire court, il est conseillé de **toujours déclarer ses variables** (on évite de surcharger le global scope, ECMA6 ...).
Leur durée de vie est différente ; une variable locale est détruite à la fin de la fonction qui la contient (Current execution context), une variable globale lorsqu'on ferme la page HTML.

Il y a bien sûr plus de subtilités que ça mais gardez à l'esprit qu'il faut utiliser var si on déclare une variable sans l'assigner et que seule une variable non déclarée peut être supprimée (ce qui est déconseillé de toute façon).

Scope

En JavaScript, les objets et les fonctions sont aussi des variables.

Le scope représente l'ensemble des variables, fonctions et objets auxquels vous avez accès à un moment donné. Il change donc au sein d'un fonction (Function scope).

Le scope remonte toujours du local vers le général, c'est-à-dire que si JavaScript ne trouve pas une variable dans son scope, il va ensuite la rechercher dans le Global et enfin générer une erreur s'il ne trouve rien.

Hoisting

Javascript déclare toujours les variables et les fonctions avant que tout code soit exécuté ; càd qu'avant toute autre chose il les déplace en début du bloc auxquelles elles appartiennent.

Il est conseillé de **toujours déclarer les variables et les fonctions au début** de leur scope.

Les conditions

Les embranchements conditionnels sont utilisés pour ***exécuter différentes actions basé sur différentes conditions***.

Les embranchements conditionnels

Très souvent, lorsque vous écrivez du code, vous voulez pouvoir exécuter différentes actions en fonction de certains critères ou décisions, c'est tout l'intérêt des conditions en programmation.

En JavaScript nous avons les différentes conditions suivantes possibles :

- Il est possible d'utiliser **if** pour spécifier un bloc de code à exécuter uniquement si une condition est vraie.
- Il est possible d'y ajouter **else** pour spécifier qu'un second bloc de code doit être exécuté si la même condition est fausse.
- Il est possible d'utiliser **else if** pour spécifier d'autres conditions à tester uniquement si la première condition est fausse.
- Enfin, il existe le mot-clé **switch** pour regrouper de nombreux blocs de code mutuellement exclusifs.

If ... then ... else ...
Notez l'espace entre else et if (et non pas elseif comme en PHP par exemple).

```
if(money <0){
    status ="I'm broke...";  }
elseif(money <10000){
    status ="Could be worse...";
}else{
    status ="I'm rich!";
```


```
}
```

Opérateur ternaire

Une écriture compacte d'un If ... then, utile pour des déclarations conditionnelles de variables par exemple.

```
varadmissibleAtl3 =(sex =="F")?true : false;
```

Switch

Une manière simple de parcourir de nombreux choix.

Notez qu'un switch permet d'avoir plusieurs cas pointant vers le même bout de code et qu'il utilise une **comparaison stricte**(===).

```
switch (newDate().getDay()){

        case1:


        case2:
        case3:
        default:

                text ="Looking forward to the Weekend
                ";
                break;

        case4:
        case5:

                text ="I'm tired and bored";
                break;

        case0:
        case6:
                text ="Time to play : )";

}
```

Les opérateurs logiques

- == (différent de =), !=
- ===, !== : comparaison de valeur et de type
- >, >=, <, <=

- && (And)
- || (Or)
- ! (Not)
    - toggle = !toggle // true devient false et false devient true.

JavaScript, comme tous les langages, s'arrête au premier false dans une condition avec des && (and), ce qui permet de d'abord tester l'existence d'une variable puis de travailler dessus sans erreur si elle n'existe pas.

JavaScript ne possède pas de XOR (exclusive OR) mais il est facilement remplaçable par une fonction.

	&& (AND)	|| (OR)	XOR
True & True	True	True	False
True & False	False	True	True
False & True	False	True	True
False & False	False	False	False

Exercice : Créez une condition avec deux paramètres qui remplace XOR.

```
vara =true;      // Testez 4x avec true, true,  false, false
varb =true;      //                true, false, false, true
if(...){
    console.log('XOR est vrai');
}else{
    console.log('XOR est faux');
};
```

Les boucles

```
// Boucle FOR
for(vari =0;i <10;i++){...};

// Boucle FOR optimisée
for(vari =0,len =myArray.length;i <len;i++){...};

// Boucle FOR IN pour itérer les propriétés d'un objet
for(prop inobj){
```


```
      /   prop
      /   obj[prop]
};

// Boucle
vari =0;
while(i <10){
      // ...
      i++; // Sans cette incrémentation, la boucle est infinie
};

// Boucle exécutée au moins une fois
do{
      // ...
      i++;
} while(i <10);
```

Les navigateurs ont tous un temps d'exécution maximum pour JavaScript.

Break, continue & label

Break et continue donnent un contrôle sur les itérations d'une boucle.

Label, lui, précise un endroit dans le script vers lequel il est possible de pointer. Il doit être déclaré avant break ou continue.

- **break** : sort de la boucle
  - break label : peut sortir de n'importe quel bloc de code (plusieurs boucles)
- **continue** : passe l'itération actuelle de la boucle

Exercice : Créez une liste à puces qui parcoure trois fois le contenu d'un tableau (avec une courte explication de getElementById & innerHTML).

Try catch

Un bloc try catch permet de 'tester' un bout de code et de récupérer l'erreur qu'il pourrait générer.

```
try{
    // Bloc de code qui pourrait générer une erreur.
}
catch(e){
    // Bloc de code pour gérer l'éventuelle erreur e.
}
Finally{
```


```
    // Bloc de code qui sera exécuté indépendamment du résultat
    du //try catch (return ou autre throw exception).
}
```

Le mot réservé **throw** permet de lancer une erreur (y compris de votre propre type).

```
throw404;
thrownewError("Invalid age");
```

Fonctions

Les fonctions (functions) ont de nombreux avantages :

- Le regroupement de code
- Améliore la lisibilité
- Réutilisation & refactoring
    - Généralisation (même code, différentes valeurs)

```
functionsum(arg1,arg2){returnarg1 +arg2 };
```

Prenez l'habitude de faire systématiquement des fonctions qui retournent toujours quelque chose (**return**) et dont le data type est consistant.

Les arguments en trop sont ignorés et les arguments non déclarés ont 'undefined' comme valeur; faites vos propres tests de validité dans la fonction via l'objet **arguments** qui est globalement similaire à un array (.length et index via [...]).

La syntaxe JavaScript des fonctions

Une **fonction** en JavaScript est définie via le mot-clé function suivi par un identifiant **unique**, suivi lui-même par des parenthèses **()**.

Les noms de fonction peuvent contenir des lettres, des nombres, le caractère souligné et le symbole dollar mais ne peut commencer par un chiffre. (même règles que les variables).

Les parenthèses peuvent inclure les noms des paramètres séparés par une virgule: (**parameter1, parameter2**, ...)

Le code qui sera exécuté par la fonction se trouve, lui, entre des accolades **{ }**

function name(parameter1, parameter2, parameter3) {
*   code to be executed*
}
Les **paramètres** de fonction sont les noms listés dans la définition de fonction.
Les **arguments** sont les **valeurs réelles** reçues par la fonction lorsqu'elle est invoquée.

Au sein de la fonction, les arguments (les paramètres) se comportent comme des variables locales.

Invocation de fonction et le mot-clé Return

Invocation de fonction

Le code entre les accolades de la fonction sera exécuté lorsque "quelque chose" **invoque** (appelle) cette fonction :

- lorsqu'un événement s'exécute (lorsqu'un utilisateur clique sur un bouton par exemple)
- lorsqu'elle est invoquée (appelée) par du code JavaScript
- Automatiquement (lorsqu'elle s'invoque elle-même)

Retour de fonction via le mot-clé Return

Lorsque JavaScript atteint le mot-clé **return**, la fonction va s'arrêter. Si la fonction a été appelée par du code, JavaScript va "retourner" à l'endroit qui l'a invoquée. Les fonctions calculent souvent une valeur de retour (**return value**). Cette valeur est renvoyée à ce qui l'a appelé :

Exemple: Multiplier un nombre par un autre et retourner le résultat :

```
var x = myFunction(4, 3);        // Appel de fonction, la valeur de retour sera stockée dans x
function myFunction(a, b) {
    return a * b;                // Retour de fonction
}
```

La valeur de x sera 12

Appel et référence de fonction

Une variable peut, bien sûr, référencer une fonction via son nom. L'appel quant à lui, se fait via les parenthèses () et d'éventuels arguments.

On appelle **High order function** toute fonction qui accepte en paramètre une ou plusieurs autres fonctions ou qui retourne une autre fonction.

Fonctions anonymes

```
varmaFonction =function(message){ alert(message); }; maFonction('
Ceci est un test'); // Affiche : Ceci est un test
```

Exercice : Créer une fonction factorielle(n) { ... } qui retourne la factorielle de n.

Closure

Le sujet est spécifique à JS et trop vaste pour une introduction mais rappelez-vous au minimum que lorsque que vous voyez le mot-clé function dans une autre fonction, la fonction interne a accès aux variables de la fonction externe (un peu comme un scope 'régional' entre le local et le global).

Exemple : http://www.w3schools.com/js/js_function_closures.asp

Les objets

Au contraire des variables de type String, Number ou Boolean, les objets peuvent contenir plusieurs valeurs sous la forme de **couples nom: valeur**.

```
varx1 ={};// Nouvel objet

varperson ={firstName: "David",
            lastName: "Collignon",
            age: 39
           };

console.log(person.firstName);
console.log(person['firstName']);
```

Objets anonymes

Un objet, comme d'autres types de données, n'a pas forcément besoin d'être nommé. C'est souvent le cas pour un objet de configuration utilisé en paramètre d'une classe.

```
$('.bxslider').bxSlider({mode:'fade', captions:true});
```

Objets courants

De nombreux objets très utiles sont directement disponibles : document, window, Math, etc.

- Objet : par exemple document
  - Propriété : par exemple .innerHTML ou .textContent
  - Méthode : par exemple .getElementById()
  - Mot-clé : this

```
document.getElementById("demo").innerHTML ="Hello World!";
```

Le mot-clé this retourne l'objet qui 'possède' le bout de code ; dans un objet ce sera donc l'objet lui-même. Nous en reparlerons en détail dans jQuery.

Assignation par valeur et par référence

En JavaScript, les data types complexes (array & object) sont assignés par référence et non par valeur. En fonction de vos besoins, il vous faudra coder un script permettant de copier votre tableau ou votre objet (Deep copy).


```
// Exemples courts de Deep copy JSON.parse(JSON.stringify(obj
))// uniq. s'il n'y a pas de fn varnewObject =jQuery.extend(
true,{},oldObject); varnewArray =jQuery.extend(true,[],
oldArray);
```

Le DOM

Qu’est-ce que le DOM?

Le DOM est un standard du W3C (World Wide Web Consortium).

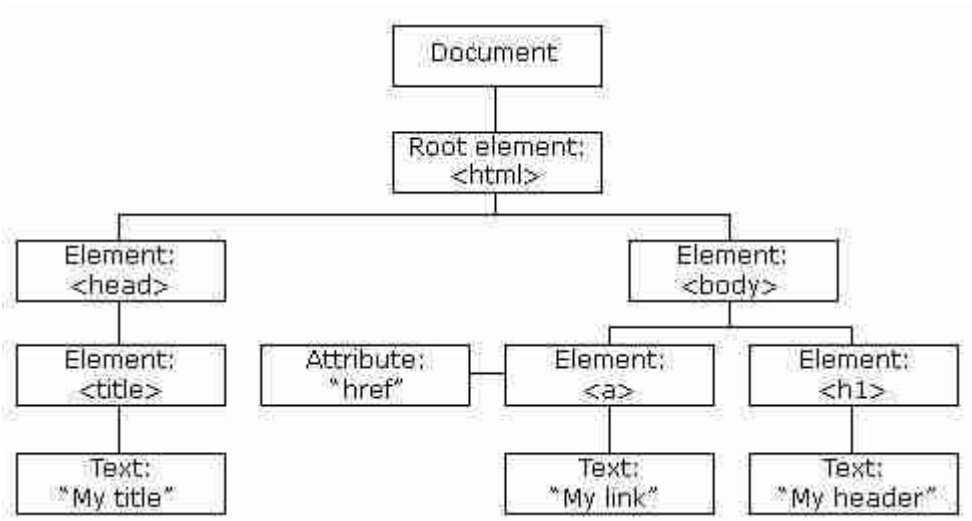
Il définit un moyen d’accéder au document:

"Le Document Object Model (DOM) du W3C est une plateforme et un interface indépendant de tout langage qui permet aux programmes et aux scripts d’accéder et de mettre à jour dynamiquement le contenu, la structure et le style d’un document."

Le standard du DOM est séparé en 3 différentes parties:

- Core DOM - le modèle standard pour tous les types de documents
- XML DOM - le modèle standard pour les documents XML
- HTML DOM - le modèle standard pour les documents HTML

JavaScript permet de parcourir, lire et modifier le Document Object Model (DOM).



Exercice : Convertir le diagramme précédent en page HTML.

L’objet document et sélectionner une partie de l’HTML

L'objet document représente l'entièreté du document HTML (le root) lorsqu'il est chargé par la fenêtre du navigateur. Il propose de nombreuses méthodes pour en cibler une partie comme par exemple :

```
varnode1 =document.getElementById("demo"); // #demo varnode2 =
document.querySelector("p"); // Le 1er p trouvé
```

getElementByID et querySelector renvoient respectivement l'id correspondant et uniquement la première occurrence trouvée ou **null** dans les cas contraires.

```
varnodeList1  =document.getElementsByClassName("demo");  //  .demo  var
nodeList2  =document.getElementsByTagName("p");//  Tout  les  p  var
nodeList3 =document.querySelectorAll("p"); // Tout les p
```

Ces méthodes renvoient une liste de nœuds (Node list) qui possède la propriété **.length**et dont les index sont accessibles via les crochets**[ ]**,il est donc possible de la parcourir via une boucle for. Néanmoins, ce n'est pas un tableau (Array).

Gardez en mémoire que, pour des sélecteurs CSS complexes, la méthode .querySelector ne vaut toujours pas des librairie spécifiques comme jQuery ni en terme de performance, ni en terme de facilité d'utilisation. De plus, depuis sa conception, certaines personnes (comme John Resig) émettent des critiques sur son implémentation.

Lire & changer l’ HTML

Qu’est-ce que le DOM en HTML?

En voici une bonne explication :

On peut modifier rapidement le contenu et les attributs d'une balise HTML avec .innerHTML (en Read & Write) et un Getter/Setter pour les attributs.

```
// Get & Set
varhref        =document.getElementById("myLink").getAttribute("href");
document.getElementById("myLink").setAttribute("href",newValue);     //
Tester l'existence
```


```
varhasH =document.getElementById("myLink").hasAttribute("href"); //
Suppression document.getElementById("myLink").removeAttribute("href");

// Injecter de l'HTML document.getElementById("demo").innerHTML ="
Nouveau texte"; // Ajouter un contenu de type texte document.
getElementById("demo").textContent ="Nouveau texte";
```

Notez la différence entre .innerHTML et .textContent, le premier est parsé et accepte donc de l'HTML et le deuxième non. **TextContent** sera plus **rapide** et surtout plus **sécurisé**.

Changer le CSS

Toutes les propriétés CSS sont accessibles via la syntaxe pointée sous la propriété style avec les particularités suivantes :

- Les propriétés sont écrites en Lower Camel Case (fontsize → fontSize)
- Les valeurs sont toujours des Strings (par exemple non pas 250 mais bien "250px")
- La nouvelle valeur sera ajoutée en **style inline**!
- La position se trouve grâce à .offsetTop et .offsetLeft (ni right, ni bottom)
- Les largeurs et hauteurs totales (padding, border) via .offsetWidth et .offsetHeigth

```
varelement =document.getElementById("demo"); element.style.
backgroundColor ="green";// camel Cased element.style["
backgroundcolor"]="green"; // CSS standard

varelement =document.getElementById("demo"); element.className =
element.className +"otherclass";// espace
```

Exercice : Comment retirer l'une des classes d'un élément?

JavaScript récupère habituellement le style inline. Si vous voulez les styles CSS résultant d'une feuille de style externe ou de la balise <style>, utilisez .getComputedStyle() :

```
varstyle =window.getComputedStyle(document.querySelector('nav'),null); var
bgc =style.getPropertyValue('backgroundcolor');//rgb(255, 191, 0)
```

Changer le DOM

Quelques méthodes pour créer et insérer des balises HTML dans le DOM.

- **document.createElement("htmlTag")** - Crée une balise HTML
- **document.createTextNode("Hello world")** - Crée un contenu texte
- **.removeChild(childToBeRemoved)** - Supprime un élément enfant
- **.appendChild(newContent)** - Rajoute un nouvel élément à la fin du parent
- **.insertBefore(newElement, currentElement)** - Rajoute un nouvel élément
- **.replaceChild(newElement, currentElement)** - Remplace un élément

Quelques méthodes utiles pour parcourir le DOM.

- **.parentNode -** Cible la node parent
- **.children[index]** - Cible les enfants via un index commençant à 0
- **.nextElementSibling** - cible la prochaine node 'soeur' (qui a le même parent)
    - Ne pas confondre avec .nextSibling qui retourne aussi les espaces entre les nodes et les commentaires!
- **.previousElementSibling** - Cible la précédente node 'soeur'

- ...

Essayez vous-même!

Les éléments du DOM en JavaScript :

Changer le Contenu HTML : http://JavaScript HTML DOM - Changing HTML

Changer le CSS:

Liste complète des propriétés et méthodes :
http://www.w3schools.com/jsref/dom_obj_all.asp

Exemples - W3Schools [1][3]

L’objet Screen

L'objet screen donne des informations sur l'écran de l'utilisateur. Ces informations ne sont **pas accessibles** par un langage de script **côté serveur**.

- **screen.width**& **screen.height**
  - **screen.availHeight**& **screen.availWidth** (moins la Windows taskbar)
- **screen.colorDepth**

L’objet Navigator

L'objet navigator donne des informations sur le navigateur de l'utilisateur.

- **navigator.userAgent** - Le nom complet du navigateur
- **navigator.platform** - L'OS de l'utilisateur
- **navigator.onLine** - Si l'utilisateur est online ou offline
- **navigator.language** - La langue d'interface du navigateur
- **navigator.geolocation** - La géolocalisation après accord de l'utilisateur
- **navigator.cookieEnabled** - Si l'utilisateur accepte les cookies
- ...

Exemple : http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_nav_geolocation

Les événements

Il est possible de lier n'importe quel événement (clic, double clic, survol, erreur, redimensionnement...) à un élément du DOM via la syntaxe suivante:

```
element.addEventListener(event, function, useCapture);
```

```
document.getElementById("myBtn").addEventListener("click",
doStuff);

function doStuff(ev){...}
```

Le nom du paramètre événement est une chaîne de caractères qui ne tient pas compte du 'on' des attributs HTML (exemple : <a href="" onClick="..." /> → click).

Liste complète sur W3Schools : http://www.w3schools.com/jsref/dom_obj_event.asp

En simple, les événements HTML sont les différentes "choses" qui peuvent se produire sur chaque élément HTML.

Lorsque JavaScript est utilisé dans des pages HTML, JavaScript peut "réagir" à ces événements.

Les événements sont des actions ou des occurrences qui se produisent dans le système dans lequel vous programmez, et dont le système vous informe en temps réel de manière à ce que vous puissiez y répondre de la manière appropriée. Par exemple, si l'utilisateur clique sur un bouton sur un site web, vous pourriez décider d'y répondre en affichant un message d'information.

Dans le cas du web, les événements sont enclenchés dans la fenêtre du navigateur et ont tendance à être, en général, attachés à un élément spécifique de celle-ci (un élément comme un div, un jeu d'éléments comme des liens, document HTML tout entier ou la fenêtre du navigateur). Il y a de nombreux types d'événements différents comme par exemple:

- L'utilisateur cliquant ou survolant avec sa souris un certain élément.

- L’utilisateur pressant une touche du clavier.
- L’utilisateur redimensionnant ou fermant la fenêtre du navigateur.
- Une page web qui finit de se charger.
- Une soumission d’un formulaire.
- Une vidéo qui joue, est en pause ou arrive à la fin.
- Une erreur qui se produit suite à du code erroné.

Chaque événement a un gestionnaire d’événements qui est un bloc de code généralement défini par le développeur qui sera exécuté uniquement lorsque l’événement se produira. Cela s’appelle ajouter un gestionnaire **d’événement**. Notez que l’on les appelle également des écouteurs d’évènements **(Event Listeners)** — ils sont interchangeables dans notre cas, bien que strictement parlant, l’écouteur attend que l’événement se produise et le gestionnaire est le code correspondant qui sera exécuté.
Un simple exemple : un bouton HTML ( ) qui, lorsqu’il sera pressé, changera de manière aléatoire la couleur de fond.

```
1 <button>Change color</button>
```

The JavaScript looks like so:

```
var btn = document.querySelector('button');

function random(number) {
  return Math.floor(Math.random()*number);
}

btn.onclick = function() {
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255)
  document.body.style.backgroundColor = rndCol;
}
```

Dans ce code, nous stockons une référence au bouton dans la variable btn en utilisant une méthode. Nous définissons également une fonction qui retourne un nombre aléatoire. La troisième partie est le gestionnaire d'événement. La variable btn pointe vers une élément <button> et ce type d'objet a un certain nombre d'événements disponibles. Il est donc possible d'y lier un écouteur et du code personnel. Nous écoutons l'événement click via le gestionnaire onclick auquel est lié une fonction anonyme qui contient notre code qui génère un code couleur RGB aléatoire et met à jour la propriété background-color du <body>.

Ce code sera exécuté chaque fois que l'événement se déclenche sur le <button>, c.-à-d. chaque fois que l'utilisateur clique dessus.

Code et résultat:

Propriétés des gestionnaires d’événements

Revenons à l’exemple précédent :

```
var btn = document.querySelector('button');

btn.onclick = function() {
  var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(25
  document.body.style.backgroundColor = rndCol;
}
```

La propriété est la propriété du gestionnaire d'événements utilisée dans cette situation. Il s'agit essentiellement d'une propriété comme toute autre disponible sur le bouton (par exemple, ou), mais c'est un type spécial - lorsque vous l'établissez comme un code, ce code sera exécuté lorsque l'événement se déclenche sur le bouton.

Gestion de l’événement

L'objet Event représente n'importe quel événement du DOM. Comme il est passé en argument, nous pouvons l'utiliser dans notre fonction qui gère l'événement.

Il possède de nombreuses propriétés et méthodes dont ...

- **event.target** - L'objet qui a émis l'événement
- **event.currentTarget** - Renvoi l'élément auquel l'événement a été attaché
- **event.preventDefault()** - Annule le comportement normal
- **event.stopPropagation()** - Annule la propagation de l'événement.

Capturing & Bubbling

La **capture** est la phase **descendante** (de l'outer element vers l'inner element) et le **bubbling** la phase **ascendante** (Comme un pêcheur qui plonge et remonte).

La valeur par défaut de useCapture est false et IE9 ne supporte que l'event bubbling.

Supprimer des event handlers

Il suffit d'utiliser .removeEventListener() mais il n'est pas possible de supprimer des fonctions anonymes. De plus, chaque event handler doit être retiré séparément ; un événement en bubbling et un autre en capturing sur le même événement sont bien deux eventListener différents.

```
vare =document.getElementById("myDIV") e.
addEventListener("mousemove",myFunction); e.
removeEventListener("mousemove",myFunction);
```

Quizz

Voici un petit quizz dont vous connaissez dorénavant toutes les réponses 🏝
le corrigé se trouve à la fin. Ne trichez pas ;)

Questions

Introduction

1. Le premier réflexe à avoir lorsqu'un programme ne fonctionne pas est de consulter les messages d'erreurs. Comment faire apparaître la console sous Firefox et Chrome ? Cochez les bonnes réponses.

- F12 (Chrome et Firefox)
- Open Menu > Developer > Web Console (Firefox)
  Open Menu > More Tools > Developer Tools > Onglet Console (Chrome)
- Ctrl + Shift + K (Firefox)
  Ctrl + Shift + I (Chrome)
- Ctrl + C (Chrome et Firefox)

2. Il est possible de déclarer une variable avec ou sans le mot clé 'var'. Quelle est la différence ?

- Aucune différence.
- Une variable qui est déclarée via 'var' est locale, l'autre est globale. Il est conseillé d'utiliser var pour ne pas surcharger le 'global scope'.
- Une variable qui est déclarée via 'var' est globale, l'autre est locale. Il est conseillé de ne pas utiliser var pour ne pas surcharger le 'global scope'.

3. Pourquoi utiliser des commentaires ?

- Il est crucial d'expliquer certains points d'un programme. Pour soi-même lorsqu'on reprend un projet après un certain laps de temps et lorsqu'on travaille en équipe.
- Les commentaires sont une perte de temps, personne n'en fait de toute façon.
- C'est un moyen facile de temporairement désactiver un bout de code sans l'effacer.

Conditions

1. Y a-t ’il une différence entre une structure 'if else' et deux 'if' consécutifs ayant des conditions opposés ?

- Non mais la première écriture (if else) est privilégiée car beaucoup plus lisible et offre moins de risque d'erreur.
- Oui, ces deux codes sont différents.

2. Comment itérer à travers tous les éléments d'un tableau d'une longueur inconnue ?

- Via la propriété .length du tableau.
- Via la méthode .length() du tableau.
- Via la fonction count()

3. *Quelle amélioration le code*
for (var i = 0, len = a.length; i < len, i++) { /* ... */ }
apporte-t-il par rapport à
for (var i = 0; i < a.length, i++) { /* ... */ }
?

- Aucune
- La variable 'len' n'est calculée qu'une seule fois et, non pas, à chaque itération.
- Le nombre d'itération est différent.

Fonctions

1. L'ordre des arguments d'une fonction 'division' qui retournerait le résultat de la division de son premier argument par le deuxième est-il important ?

- Oui, le résultat serait différent.
- Non

2. Quelle est la différence entre un appel de fonction et une référence de fonction ?

- L'appel de fonction (son nom suivi de parenthèses) demande l'exécution du code contenu dans la fonction au moment de l'appel. Une référence de fonction (comme un call back par exemple) stocke temporairement le nom de la fonction a appeler au moment opportun.
- Aucune différence si la fonction n'a pas d'argument.

3. Quel est le mot clé réservé pour spécifier la valeur retournée par un appel de fonction ?

- Return
- Pass
- Exit

DOM

1. Quelles sont les méthodes qui ne renvoient que la première (ou la seule) occurrence trouvée ?

- document.querySelector()
- document.querySelectorAll()
- document.getElementById()
- document.getElementsByTagName()
- document.getElementsByClassName
- document.getElementsByName()

2. Dans le code suivant, quel argument reçoit le callback ?
document.getElementById('myID').addEventListener('click', myCallBack);

- L'événement de type 'click' qui a activé le callback.
- Aucun argument n'est passé.

3. Par quel moyen peut-on récupérer la valeur contenu dans un contrôle HTML (UI) tel qu'une balise input ?

- Via sa propriété .value qui retourne le contenu de l'attribut 'value'.
- Via sa méthode .getValue() qui retourne le contenu de l'attribut 'value'.
- Via la méthode .val() qui retourne le contenu de l'attribut 'value'.

CSS

1. Quel sélecteur CSS cible uniquement chaque paragraphe descendant direct d'un div ?

- Div > p
- Div p
- Div, p

2. Quel sélecteur CSS cible uniquement le premier des paragraphes quel que soit ses éléments HTML frères.

- p:nth-of-type(1)
- p:first-child
- p:nth-child(1)

3. Si deux sélecteurs CSS (.rouge et #bleu) ciblent le même élément, quel sera la couleur de celui-ci

- Bleu
- Rouge
- Mauve
- Cela dépend ; celui qui est déclaré en dernier.

Réponses

Introduction

1. Le premier réflexe à avoir lorsqu'un programme ne fonctionne pas est de consulter les messages d'erreurs. Comment faire apparaître la console sous Firefox et Chrome ? Cochez les bonnes réponses.

- F12 (Chrome et Firefox)
- Open Menu > Developer > Web Console (Firefox)
  Open Menu > More Tools > Developer Tools > Onglet Console (Chrome)
- Ctrl + Shift + K (Firefox)
  Ctrl + Shift + I (Chrome)

2. Il est possible de déclarer une variable avec ou sans le mot clé 'var'. Quelle est la différence ?

- Une variable qui est déclarée via 'var' est locale, l'autre est globale. Il est conseillé d'utiliser var pour ne pas surcharger le 'global scope'.

3. Pourquoi utiliser des commentaires ?

- Il est crucial d'expliquer certains points d'un programme. Pour soi-même lorsqu'on reprend un projet après un certain laps de temps et lorsqu'on travaille en équipe.
- C'est un moyen facile de temporairement désactiver un bout de code sans l'effacer.

Conditions

1. Y a-t ‘il une différence entre une structure 'if else' et deux 'if' consécutifs ayant des conditions opposés ?

- Non mais la première écriture (if else) est privilégiée car beaucoup plus lisible et offre moins de risque d'erreur.

2. Comment itérer à travers tous les éléments d'un tableau d'une longueur inconnue ?

- Via la propriété .length du tableau.

3. Quelle amélioration le code
for (var i = 0, len = a.length; i < len, i++) { /* ... */ }
apporte-t-il par rapport à
for (var i = 0; i < a.length, i++) { /* ... */ }
?

- La variable 'len' n'est calculée qu'une seule fois et, non pas, à chaque itération.

Fonctions

1. L'ordre des arguments d'une fonction 'division' qui retournerait le résultat de la division de son premier argument par le deuxième est-il important ?

- Oui, le résultat serait différent.

2. Quelle est la différence entre un appel de fonction et une référence de fonction ?

- L'appel de fonction (son nom suivi de parenthèses) demande l'exécution du code contenu dans la fonction au moment de l'appel. Une référence de fonction (comme un call back par exemple) stocke temporairement le nom de la fonction a appeler au moment opportun.

3. Quel est le mot clé réservé pour spécifier la valeur retournée par un appel de fonction ?

- Return

DOM

1. Quelles sont les méthodes qui ne renvoient que la première (ou la seule) occurrence trouvée ?

- document.querySelector()
- document.getElementById()

2. Dans le code suivant, quel argument reçoit le callback ?
document.getElementById('myID').addEventListener('click', myCallBack);

- L'événement de type 'click' qui a activé le callback.

3. Par quel moyen peut-on récupérer la valeur contenu dans un contrôle HTML (UI) tel qu'une balise input ?

- Via sa propriété .value qui retourne le contenu de l'attribut 'value'.

CSS

1. Quel sélecteur CSS cible uniquement chaque paragraphe descendant direct d'un div ?

- Div > p

2. Quel sélecteur CSS cible uniquement le premier des paragraphes quel que soit ses éléments HTML frères.

- p:nth-of-type(1)

3. Si deux sélecteurs CSS (.rouge et #bleu) ciblent le même élément, quel sera la couleur de celui-ci

- Bleu

Labo code : Devinez un chiffre aléatoire

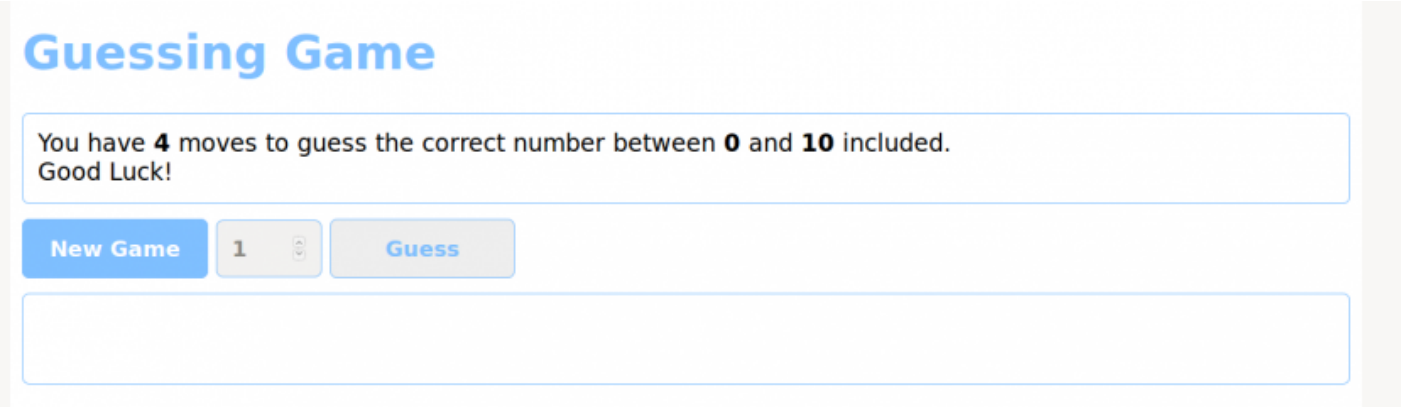
Maintenant que vous comprenez les bases d'HTML, de CSS et de JavaScript, il est temps de s'amuser avec notre premier petit projet...
Dans cette leçon, vous allez recréer un jeu de devinette. Pour cela, il faudra créer :

1. Une page HTML dans laquelle vous mettrez tous les éléments nécessaires au jeu.

2. Un fichier CSS pour restyler votre jeu de la manière que vous voulez ;-)

3. un fichier JavaScript qui contiendra toutes les fonctions nécessaires pour le jeu

Voilà à quoi vous devriez arriver (mais pas spécialement le même visuel):



Vous trouverez au besoin une explication étape par étape sur la page suivante.

On commence…

1. Nous aurons besoin au moins d'un contrôle numérique (numeric stepper) et un bouton pour tester le résultat mais aussi une forme de 'conteneur' dans lequel nous allons écrire les résultats même s'il commence vide ou invisible.

Posez-vous la question si un bouton 'Nouvelle partie' est nécessaire ou utile, à la fois du point de vue du joueur et du codeur. Vous pouvez par exemple modifier votre code pour permettre l'abandon d'un partie en cours.

2. Essayez de penser à toutes les étapes nécessaires pour réaliser ce mini jeu. Ce sera le début de réflexion des blocs de codes nécessaires. Si un problème vous semble difficile à résoudre, vous devrez probablement le couper en de plus petites opérations plus faciles à résoudre.

Si vous réutilisez plusieurs fois la même logique ou le même bout de code, il est vivement conseillé de le transformer en fonction, surtout s'il est possible ou nécessaire d'en changer les paramètres.

Utilisez toujours des variables plutôt que des valeurs fixes (hardcoded) de manière à pouvoir facilement changer certains paramètres du jeu comme le nombre de coups permis par exemple. Cela augmente la lisibilité, la maintenance et rend le code plus configurable.

Commentez toujours votre code (en expliquant bien la logique derrière votre code et non pas le code lui-même) pour vous, votre vous futur ;) et vos collègues.

3. getRandomIntegerBetween() est un simple exemple de fonction qui améliore à la fois la lisibilité et rend un bout de code réutilisable.

Prenez en compte le fait que demander quelque chose de réellement aléatoire à une machine n'est pas quelque chose de facile en informatique. Les ordinateurs sont des machines déterministes, ce qui signifie que chaque fois que vous leur posez la même question, vous recevrez la même réponse. En fait, de telles machines sont spécifiquement programmées pour éliminer tout facteur incontrôlable. Regardez pseudo random dans Google si vous voulez en savoir plus.

Dans checkResponse(), nous voyons que return peut aussi être utilisé pour sortir d'une fonction sans rien renvoyer, évitant ainsi un test inutile pour voir si le jeu est perdu alors qu'il est déjà gagné.

Enfin, nous voyons que les technologies web comme JavaScript sont fortement centrées sur l'accès et la modification du DOM HTML avec des outils tels que document.getElementById('userGuess') et resultP.innerHTML. Chacunes de ces technologies relativement simples fonctionne ensemble pour créer quelque chose de plus complexe.

Page suivante, quelques indices pour la structure et les fonctions…

STRUCTURE ET FONCTIONS

```
// Retourne un entier entre min et max inclus
    function getRandomIntegerBetween(min, max)

// Active l'interface utilisateur pour une nouvelle partie
    function activateUI()

// Game over: désactive l'interface utilisateur
    function deactivateUI()

    function init(){
        // Reset au maximum
        // Vide les logs
        // Choisir un nouveau chiffre aléatoire
        // Triche ;)
        activateUI();
    }

// Vérifie la réponse
function checkResponse()

    // Nombre de coups disponibles avant le game over

    // Vérifier le chiffre choisi par le joueur

    // Montrer les règles

    // Loger les réponses de l'utilisateur
```

// Gérer l’interface utilisateur

// Quelle est la manière optimale de jouer à ce jeu ?

Références & Ressources

JAVASCRIPT

[EN] JavaScript

– -

[EN] JavaScript Style Guide

JavaScript for the Total Non-Programmer

Defer & async [EN]

-

Modernizr

-

What is the function of the var keyword and when to use it ?

[EN] The infamous 'this' keyword

-

[EN] ECMA 6 The future of JavaScript

http://es6-features.org/#Constants

The JavaScript Source is an excellent JavaScript resource with tons of "cut and paste" **JavaScript examples for your Web pages**. All for free!

HTML & CSS

HTML5 Tutorial

CSS3 Tutorial

Et bien plus encore sur Google, YouTube et tout le web!

MERCI à David Collignon, pour son aide précieuse et ses notes de cours