HUMAN **booster**

VOTRE SOLUTION COMPETENCE

Javascript

Auteur / formateur : Alexandre Beaugrand - alex.beaugrand@gmail.com

Objectifs et prérequis

- **Objectif**
  - Prendre en main javascript pour en comprendre la syntaxe, agir sur vos pages web, les rendre dynamiques et interactives
  - Connaitre de nouvelles syntaxes ES6
  - Utiliser AJAX
  - Utiliser jQuery
  - Avoir une introduction aux APIs HTML5
- **Prérequis**
  - Bonnes connaissances du HTML
  - Connaissances algorithmiques

Plan de cours

1) Introduction

2) Syntaxe

3) Manipulation du DOM

4) Évènements et données

5) Gestion de formulaires

6) jQuery

7) Introduction aux APIs HTML5

Chapitre 1
Introduction

Présentation

- Javascript à été créé par Brenden Eich pour Netscape

- Ne pas confondre JavaScript avec JAVA ! JAVA est un langage différent de JavaScript à tout point de vue : son usage, sa sémantique...D'ailleurs, il n'est pas interprétable nativement par un navigateur web.

ECMASCRIPT

- Il suit le standard ECMAScript, ayant pour objet de spécifier le comportement de langages de scripts (spécifications : ECMA-262 et ECMA-402). Ce standard à historiquement été implémenté par ActionScript, mais surtout par JS

- Il en existe de multiples versions : ES1… ES9. ES.Next est un nom de version dynamique qui fait toujours référence à la prochaine version d'ECMAScript.

Les langages web

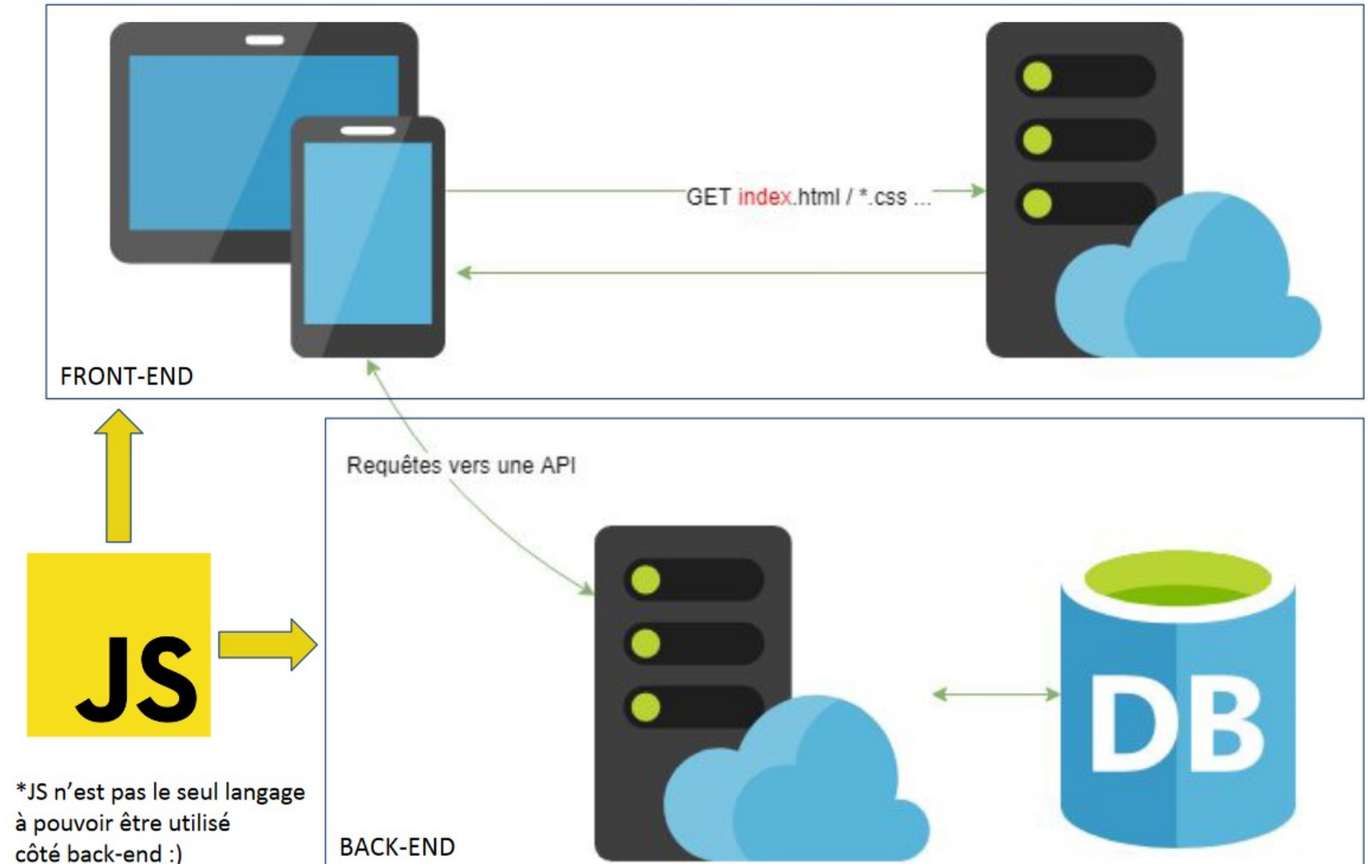
- HTML
- CSS
- Javascript
- Langages back : PHP, ASP,… et Javascript !

Éxecution

- JavaScript est un langage interprété et à typage dynamique. Cela signifie que le code n'est pas vérifié avant son exécution.
- Il est possible d'exécuter du javascript au sein d'un navigateur, ou bien en dehors du navigateur.
- Javascript peut ainsi être le langage principal du back-end, avec NodeJS

Node.JS, un programme permettant d'interpréter du JavaScript

Exemple d'architecture



Exemples d'utilisation

- Images changeant au survol
- Formulaires dynamiques
- Vérifications de formulaires
- Diaporamas
- Etc.

Moteur d'interprétation

- Javascript est interprété par un moteur
- Il peut donc y avoir des différences entre les moteurs, ce qui engendre des problèmes de compatibilité et des différences de performances suivant les moteurs
- Le “moteur V8” est sans doute le moteur d’interprétation JS le plus connu

Chapitre 2
Syntaxe

Où mettre son code ?

- Directement sur la page (dans head ou body)

```
<script>Code</script>
```

- Dans un fichier externe

```
<script src="fichier.js"></script>
```

- Dans une balise (pour écouter des evenements)

```
<a href='#' onClick='code'>Cliquez ici</a>
<a href="javascript:code">lien</a>
```

Premières instructions

- Pop-up :

```
alert("helloworld");
```

- Ecrire dans la page :

```
document.write("Helloworld");
```

- Ecrire dans la console :

```
console.log('test');
```

Le moteur JS comporte un mécanisme (ASI = “Automatic semicolon insertion”) qui ajoute automatiquement des points-virgules lorsqu’ils ne sont pas écrits explicitement par le développeur. Néanmoins c’est une mauvaise pratique de compter sur ce mécanisme car cela peut induire des comportements auxquels on ne pensait pas (par méconnaissance, en général, des règles de l’ASI). Voir ce lien pour plus de détails.

Commentaires

- Sur une ligne

```
//mon commentaire
```

- Sur plusieurs lignes

```
/* mon commentaire sur
  plusieurs lignes */
```

Variables

- Les variables ont un nom (commence par une lettre ou _, contient chiffres, lettres, _ et & sans espaces)
- Elles sont liées à des valeurs modifiables au cours du script
- On peut utiliser les mot clés var, let, ou const pour les déclarer. var et let pour des variables classiques, const pour des constantes.
- Exemple :

```
var nom = "toto";              //variable
const MESSAGE = "Bonjour";     //invariable
```

- Par défaut, il est possible d'affecter une valeur à une variable non-déclarée par var / let / const. Elle est alors implicitement déclarée dans le contexte global, quel que soit l'endroit de l'affectation.

Types

- Les variables ont un type implicite, qui n'a pas besoin d'être précisé

- Il est toutefois possible de spécifier des types
  - soit via une librairie complémentaire type Flow
  - soit en utilisant une surcouche au langage comme TypeScript

- Types primitifs
  - Number
  - String
  - Boolean
  - Object (type de Date, Array, RegExp, etc.)
  - Function
  - Undefined (variable déclarée mais non-affectée, ou bien opération/propriété/variable qui n'existe pas)

Typage, conversion, détection de type

- Conversion par fonctions :
  - parseInt() : variable vers nombre
  - parseFloat() : variable vers nombre décimal

- Conversion par casting : on entoure une valeur de parenthèse précédées du type voulu. Exemple :

```
console.log(Boolean("A"));
```

- Détection de type : typeof(). Exemple :

```
var v="test"; console.log(typeof(v));
```

Var et Let

- Par défaut, une variable déclarée avec “var” a pour portée :
  - la fonction qui contient la déclaration
  - le contexte global si la variable est définie en dehors de toute fonction

- Avec le mot clef “let” il est possible de définir une variable dont la portée sera limitée à celle du bloc dans lequel elle sera déclarée.

```
let x = 1;
if (x == 1) {
    let x = 2;      //seconde variable x qui masque le 1er x dans ce bloc
    console.log(x); //2
}
console.log(x);     //1
```

Var et Let

- Une autre différence clé est qu'il n'est pas possible de déclarer 2 fois une même variable dans le même scope avec let

```
let nom = "toto";
let nom = "tata";   //erreur car même scope
console.log(nom);

let nom = "toto";
if (true) {
    let nom = "tata";
}
console.log(nom);   //pas d'erreur car scopes différents
```

- En règle générale aujourd'hui on privilégie l'utilisation de let par rapport à var

Mode strict

- Il est possible de développer dans une “variante” du JavaScript, plus stricte. Cette variante a été introduite avec ES5.

- Ses principes fondamentaux sont :
  - de ne pas autoriser l’usage de variables non déclarées
  - de lever une exception si une affectation est incorrecte (exemple : nommer une variable par un mot clef du langage)
  - d’empêcher la duplication de propriétés dans un objet (on ne peut plus définir d’objet ayant 2 propriétés avec même nom)
  - d’empêcher la duplication d’arguments dans une signature de fonction (idem, qui auraient le même nom)
  - facilite l’optimisation du code par le navigateur
  - interdit la déclaration de fonctions n’importe où dans le code (comme dans un bloc conditionnel)

Mode strict

- Le mode strict peut s'appliquer :
  - globalement sur l'ensemble de notre fichier JS. Il faut donc indiquer en première ligne :

```
"use strict";                //mode strict
msg = "Mode strict !"        //msg n'est pas précédée de var ou let
console.log(msg);            //génère une erreur
```

  - dans le scope d'une fonction :

```
function pasStrict() {
    msg1 = "fonction pas stricte"; console.log(msg1);
}
function strict() {
    "use strict";
    msg2 = "fonction stricte"; console.log(msg2);
}
pasStrict();    //ok
strict();       //erreur
```

Hoisting

- On peut remarquer que lorsque l'on déclare une variable après l'avoir appelée, on ne génère pas d'exception

```
console.log(nom);
var nom = "toto"; //ce code est executé sans lever d'exception
```

- Le "hoisting" ("remontée de variables") est un des principes du langage. Les déclarations des variables sont traitées avant n'importe quelle autre instruction. Elles sont mêmes "remontées" au début de leur contexte d'exécution (de leur portée)

```
console.log(nom);
var nom = "toto";

//est implicitement traité comme :
var nom;
console.log(nom);
nom="toto";
```

Opérateurs

- Arithmétiques :

+ - / % * % ++ --

- D'assignation :

= += -= *= /= %=

- De concaténation :

+

Template Strings (ES6)

- En fonction des cas, utiliser l'opérateur + pour concaténer des chaînes peut être fastidieux... Depuis ES6, la notion de template string nous permet d'en finir avec l'enfer des +
  - Elles permettent d'injecter des variables, et des instructions javascript simples
  - Elles permettent aussi de passer à la ligne

```
let prenom = "toto";
let messages = 1;

console.log(`Bonjour ${prenom}
Comment allez-vous ?
Vous avez ${++messages} messages.`);
```

Opérateurs

- Comparaisons :

== === != < > >= <=

- Logiques :

&& || !

Structures de contrôle

- If / If, else / If, else if, else :

```
if (condition1){
    code 1
}else if (condition2){
    code 2
}else{
    code 3
}
```

- Opérateur ternaire :

```
let variable=(condition)?valeur1:valeur2;
```

Structures de contrôle

- switch :

```
switch(variable){

   case valeur1:
      code 1
      break;

   case valeur2:
      code 2
      break;

   default:
   code 3

}
```

Boucles communes

- For :

```
for (var i=0;i<=5;i++){
     console.log(i);
}
```

- While :

```
var i=0;
while (i<=5){
   console.log(i);
   i++;
}
```

Break et continue

- Break, au sein d'une boucle, en fait sortir
- Continue, au sein d'une boucle, passe à l'itération suivante

```
for (var i=0;i<=10;i++){
   if (i==3){
            continue;
   }
   document.write(i+'<br/>');
}
```

Fonctions

- Pour définir une fonction :

```
function saluer(){
     console.log("Bonjour");
}
```

- Pour invoquer une fonction :

```
saluer();
```

- Lorsque l'on appelle une fonction (quelque soit la méthode) ; le moteur JavaScript ajoute automatiquement (et implicitement) 2 paramètres : this et arguments

Fonctions fléchées (ES6)

- En javascript, il est possible de définir des fonctions anonymes :

```
bouton.onclick = function(e){
    console.log(e);
}
```

- En ES6 on peut déclarer ces fonctions plus rapidement :

```
bouton.onclick = (e) =>{
    console.log(e);
}
```

- Ces fonctions n'engendrent pas de création implicite de this ou arguments

Paramètres de fonctions

- On peut passer des paramètres à une fonction:

```
function saluer(prenom,nom){
    console.log("Bonjour "+prenom+" "+nom);
}
```

- On l'appelle en lui donnant le même nombre de paramètres :

```
saluer("Paul","Dupont");
```

- On peut passer des valeur par défaut en plaçant un test en début de fonction :

```
if ( nom === undefined ) {var nom = "";}
```

- Mais en ES6 on peut passer des valeurs par défaut en signature :

```
function saluer(prenom,nom=""){…}
```

Portée des variables

- Un fonction à accès aux variables du programme, mais le programme n'a accès qu'aux variables déclarées au sein d'un fonction NON précédées de "var" ou "let"

```
var nom = "toto";
function saluer() {
   a = "a";
   var b = "b";
   console.log("Bonjour " + nom);
}
saluer(); //bonjour toto
console.log(a); //a
console.log(b); //erreur (b is not defined)
```

Instanciation d'objets

- On peut créer des objets prédéfinis avec le mot clé new et le type de l'objet, selon la forme :

```
let nom_variable = new Type(param_optionnel);
```

- Exemples :

```
let v = new Number(4.05); // équivaut à var v = 4.05
let v = new String("test"); // équivaut à var v = "test"
```

- On peut utiliser ses propriétés et ses méthodes avec le point

```
let v = new String("test");
alert(v.length); //accès à la propriété length
document.write(v.bold()); //utilisation de la méthode bold()
```

Création d'objets

- On peut créer nos propres objets standards regroupants propriétés et méthodes, avec le mot clé new et le type Object :

```
var o=new Object();
o.propriete='valeur1';
o.methode=function(){alert('ok')};
var mafonction=function(){alert('ok2')};
o.methode2=mafonction;

document.write("Propriété : "+o.propriete);
o.methode();
o.methode2();
```

Création d'objet JSON

- On peut aussi déclarer des objets en utilisant la notation JSON :

```
var o={
    propriete:'valeur1',
    methode:function(){alert('ok')}
}

//utilisation
alert(o.propriete);
o.methode();
```

Destructuring d'objets (ES6)

- Le destructuring permet de copier les propriétés d'un objet dans de nouvelles variables

```
let voiture = {
    marque: "Fiat",
    modele: "500",
    annee: 2010
};

let { marque, modele } = voiture;       //simple extraction
console.log(marque + " " + modele);

let { annee: anneeDeSortie } = voiture; //extraction + renommage
console.log(anneeDeSortie);

let { couleur = "rouge" } = voiture;    //valeur par défaut si la prop. n'existe pas
console.log(couleur);
```

Boucle for in

- La boucle "for in" permet de parcourir les propriétés d'un objet :

```
for (variable in objet){
    code
}
```

- Exemple :

```
var personne = { prenom: "Jack", nom: "Sparrow" };
for (var i in personne) {
    console.log(personne[i]);
}
```

Classes

- En ES5, il n'existe pas de mot clé class, mais on peut s'en approcher avec le mot clé function

```
function maClasse(){
    //propriété
    this.nom = "Monsieur";
    //méthode
    this.saluer = function(){ alert("Bonjour "+ this.nom);  }
}
```

- Utilisation :

```
var o = new maClasse();  //création d'un objet maClasse
alert(o.nom); //accès à la propriété nom
o.saluer();  //utilisation de la méthode saluer()
```

This

- Le mot clé this sert à faire référence à l'objet qui invoque la fonction « classe »

- Dans l'exemple suivant, this fait référence à l'objet o

```
function maClasse(){
        this.nom = "Monsieur";
        this.saluer = function(){ alert("Bonjour "+ this.nom);  }
}
var o = new maClasse();
```

Paramètres et constructeur

- On peut passer des paramètres à notre classe, qui agit alors comme un constructeur en initialisant des propriétés

```
function maClasse(nom_){
     this.nom = nom_;
     this.saluer = function(){ alert("Bonjour "+ this.nom);  }
}
```

- Instanciation de l'objet

```
var o = new maClasse("Monsieur");
```

Extensions d'objets prédéfinis

- Il est possible d'ajouter des propriétés et des méthodes à des objets après qu'il aient été instanciés

- Exemple :

```
function Cercle(){ }
var petit_cercle=new Cercle();
var grand_cercle=new Cercle();
petit_cercle.pi=3.14159; //extension
alert(petit_cercle.pi);
alert(grand_cercle.pi);
```

Prototype

- Tous les objets disposent d'un prototype, qui permet de redéfinir leur « type ». On peut ainsi par ex. ajouter des propriétés et des méthodes à l'ensemble des objets d'un type donné :

```
Type.prototype.nom="toto";
```

```
function Cercle(){ }
var petit_cercle=new Cercle();
var grand_cercle=new Cercle();
Cercle.prototype.pi=3.14159;
alert(petit_cercle.pi+" "+grand_cercle.pi);
```

Pour résoudre un appel à une fonction ou une propriété sur un objet, le moteur JavaScript regarde dans l'ordre :

- si l'objet implémente la méthode ou déclare la propriété directement
- si ce n'est pas le cas, accède au prototype de son constructeur et regarde s'il trouve ce qui est demandé
- si ce n'est pas le cas, accède au prototype du constructeur parent de l'objet
- ainsi de suite jusqu'à remonter au prototype Object.

Classes (ES6)

- La POO prototypale n'est pas très courante. Beaucoup de développeurs s'y perdent. ES6 apporte les mots clés class et extends pour créer des "classes" "à la manière" de JAVA ou C. Le code est traduit selon la philosophie JavaScript par le moteur d'interprétation

```
class Animal {
    constructor(categorie) {
        this.categorie = categorie;
    }
}
class Chat extends Animal {
    constructor(nom) {
        super("mammifère");
        this.nom = nom;
    }
    decrire() {
        console.log("Il s'agit d'un " + this.categorie + " nommé " + this.nom);
    }
}
let chat1 = new Chat("Félix");
console.log(chat1.nom);
chat1.decrire();
```

Objet String : chaine de caractères

- Les objets ont des propriétés et des méthodes
- Déclaration d'un objet String

```
var txt = new String();
```
ou
```
var txt="test";
```

- Exemple de propriété

```
document.write(txt.length);
```

- Exemple de méthode

```
document.write(txt.toUpperCase());
```

- Référence des Strings
  http://www.w3schools.com/jsref/jsref_obj_string.asp

Objet String : chaine de caractères

- Pour str et str2 étant deux chaines de caractères, i, et j entiers :
- str.length // le nombre de lettres
- str.charAt(i) // caractère à la position i (1ère lettre = position 0)
- str.indexOf(str2) // 1ère position de str2 ou -1 si non trouvée
- str.lastIndexOf(str2) // idem mais avec la dernière position
- str.toLowerCase() // str en minuscules
- str.toUpperCase() // str en majuscules
- str.concat(str2) // str concaténée avec str2
- str.substring(i,j) // sous chaine de str, de i inclus à j exclus
- str.substr(i,j) // sous chaine de i jusqu'à j caractères
- str.split(str2) // tableau de str scindé à chaque str2

Objet Regexp

- Les expressions régulières sont des chaines codées définissant un motif, c'est à dire décrivant d'autres chaines

- Elles sont encadrées par des slashs suivis éventuellement d'un « modifier » : `/expression/modifier`

signe	description	signe	description
^dans /^abc/	Commence par abc	\S dans /\S/	Tout sauf espaces blancs
$ dans /abc$/	Fini par abc	\w dans /\w/	Tout signe alphanum. et _
* dans /abc*/	abc suivi de 0 ou plus c	\W dans /\W/	Inverse du précédent
+ dans /abc+/	abc suivi de 1 ou plus c	\b dans /\ba\b/	Toutes les lettres a seules
. dans /abc./	abc suivi d'un caractère	\d dans /\d.\d/	Toutes chiffre entier
\s dans /\s/	Tout espace blanc	\D dans /\D.+\D/	Caractères sans chiffres
- Dans a-z	Lettres de a à z	[ ] dans [a-zA-Z]	Minuscules ou majuscules

- Exemple de motifs : `/Mamarque/g`
  `/^\w+@\w+\.\w+$/g`

Objet Regexp

- Les motifs des expressions régulières sont plus ou moins précis

- Exemple de détection d'e-mail simple :
- `/^\w+@\w+\.\w+$/g`

- Plus précis :
- `/^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$/g`

- etc.

Pratique : Expressions régulières

- Il existe des méthodes de l'objet String et de l'Objet RegExp qui utilisent les expressions régulières. Pour str et str2 = chaines et reg = expression régulière :
- str.search(reg) // position de la 1ère occurrence de reg ou -1
- str.match(reg) // tableau des occurrences de reg dans str
- str.replace(reg,str2) // chaine avec les reg remplacées par str2
- reg.exec(str) // renvoie le premier élément correspondant ou null
- Reg.test(str) // renvoie vrai ou faux
- Exemple :

```
var exp=/toto/g;
document.write(exp.test("bonjour toto"));
```

Référence des RegExps :
http://www.w3schools.com/jsref/jsref_obj_regexp.asp

Objet Date

- Les dates sont souvent décrites en millisecondes. Elles font toutes références au nombres de millisecondes s'étant écoulées depuis le 1er janvier 1970
- Pour la suite, définissons la variable moment =
  - "mois jour, annee heures:minutes:secondes"

- Il y a 4 façons pour déclarer une Date :
  - new Date() //maintenant (en millisecondes)
  - new Date(millisecondes) //à la date indiquée en millisecondes
  - new Date(moment) // ex : "October 13, 1975 11:13:00"
  - new Date(année, mois, jours, heures, minutes, secondes, millisecondes)

- Référence des Dates : http://www.w3schools.com/jsref/jsref_obj_date.asp

Objet Date

- Pour d=date, moment = "mois jour, annee heures:minutes:secondes" et X ayant pour valeur possibles : FullYear (année), Month(mois), Date(jour du mois), Day (jour de la semaine – 0 = dimanche), Hours (heures de 0 à 23), Minuts (minutes), Seconds (secondes), Milliseconds (millisecondes)

- On a les méthodes suivantes :
- d.getX() // retourne X
- d.setX(valeur) // modifie X
- d.getUTFX() // retourne X pour Greenwhich
- d.setUTFX() // modifie X pour Greenwhich
- Date.parse(moment) // nombre de millisec. depuis le 1/1/1970
- Date.UTC(a,m,j,h,s,m) // idem, mais paramètres différents

Objet Math

- L'objet Math à des propriétés et méthodes utiles pour des calculs

- Propriétés :
  - Math.PI //retourne PI

- Méthodes (pour x,y,n = nombres) :
  - Math.abs(x) // valeur absolue
  - Math.cos(x) // cosinus. Il existe acos pour l'inverse. sin et tan également.
  - Math.ceil(x) // arrondit supérieur. Il existe floor et round également.
  - Math.pow(x,y) // x puissance y. Sqrt retourne la racine
  - Math.random() // nombre « aléatoire » compris entre 0 et 1
  - Math.max(x,y,…,n) // valeur max. Il existe min également

- Référence des Maths :
  http://www.w3schools.com/jsref/jsref_obj_math.asp

Objet Array

- Un Array est un tableau, qui contient de multiples valeurs et tous types, manipulables par des méthodes associées à l'objet Array

- Déclarations de tableaux :
  - Déclaration 1 (simple) : `var t=new Array(19,24);`
  - Déclaration 2 (simple) : `var t=[19,24];`
  - Déclaration 3 (simple) : `var t=new Array(); t[0]=19; t[1]=24;`
  - Déclaration 4 (associatif) : `var t=new Array() t["A"] = "valeur";`
  - Déclaration 5 (imbriqués) : `var t=new Array(); t[0]=new Array('A');`

Accéder aux données d'un Array

- Accès directs (avec t = objet Array) :
  - Tableau simple : `t[0];`
  - Tableau associatif : `t["A"];`
  - Tableaux imbriqués : `t[0]['A'];`

Parcourir un tableau : for in et for of

- On peut utiliser des boucles for, while, for in, et for of.
- Dans le cas de la boucle for in, l'index pointe tour à tour sur les clé du tableau
- Dans le cas de la boucle for of, l'index pointe tour à tour sur les valeurs du tableau. Exemple:

```
var elements = new Array("e1", "e2", "e3");
for (var x in elements) {
    console.log(elements[x]);
}
for (var x of elements) {
    console.log(x);
}
```

Méthodes d'un Array

- Pour t1, t2 = Array et v1, v2 = chaines ou nombres

- t1.length // longueur de t1
- t1.concat(t2) // t1 concaténé à t2
- t1.join() // chaine avec les éléments de t1 séparés par des ,
- t1.pop() // supprime le dernier élément de t1 et le retourne
- t1.push(v1,v2) // ajoute v1 et v2 à la fin de t1
- t1.reverse() // inverse l'ordre des éléments de t1

- Référence des Arrays : http://www.w3schools.com/js/js_obj_array.asp

Destructuring d'arrays (ES6)

- Il est possible de déstructurer un tableau

```
let notes = [10, 11, 12];
let [note1, note2] = notes;
console.log(note1 + " " + note2);
```

Spread Operator (ES6)

- Un nouvel opérateur est disponible avec ES6 : le spread operator. Il s'écrit "..." et s'applique sur tout objet itérable (tableau, chaîne de caractères). Il permet de décomposer tous ses éléments.

```
let tab1 = [1, 2, 3];
let tab2 = [...tab1, 4, 5, 6];
console.log(tab2);    //1 2 3 4 5 6
```

- C'est pratique, par exemple, pour remplacer des pushs :

```
//exemple 1
let tab1 = [1, 2, 3];
let tab2 = [4, 5, 6];
tab1.push(tab2);
console.log(tab1);  // 1 2 3 array

//exemple 2
let tab1 = [1, 2, 3];
let tab2 = [4, 5, 6];
tab1.push(...tab2);
console.log(tab1);  // 1 2 3 4 5 6
```

Spread Operator (ES6)

- Attention, un tableau d'objets provoque une copie de surface : une modification d'un objet dans tab1 sera visible dans tab2.

```
let t1 = [{ nom: "Félix" }, { nom: "Matou" }];
let t2 = [...t1, { nom: "Jack" }];

// le 1er objet de t2 reste Félix : l'affectation retire la référence
// à Félix dans t1, qui n'est référencé que dans t2
//t1[0] = { nom: "Rex" };

// le 1er objet de t2 devient Rex : Félix et Matou sont référencés dans t1 et t2,
// les modifier impacte t1 et t2
t1[0].nom = "Rex";

console.log(t2);
```

Modules

- Lorsque l'on développe, il est intéressant d'isoler du code indépendant et réutilisable. Les modules ES6 permettent de faire cela. Chaque module contient du code privé et du code exposé, utilisable dans d'autres fichiers. C'est un mécanisme d'import / export. Le code d'un module est par défaut en mode strict
- Si on inclut un script en précisant que c'est un module, on peut bénéficier des mots clé import et export :

```
<!-- préciser type="module" en HTML -->
<script type="module" src="main.js"></script>
```

```
//main.js
import { saluer } from "./module.js";
saluer();
```

```
//module.js
function saluer() {
    console.log("Bonjour");
}
export { saluer };
```

Veiller à tester ces scripts en passant par un serveur.

Les modules ES6 sont surtout utilisés côté navigateur. Dans un contexte node, on utilise plutôt la syntaxe CommonJS avec les mots clefs module.exports et require. Node supporte cependant les modules ES6.

Timers

- On utilise la fonction `setTimeOut(fct,delaiEnMillisecondes)` pour appeler une fonction fct après un délai. Exemple :

- Dans head :

```
function decompte(){
    var t=setTimeout("alert('2 secondes après !')",2000);
}
```

- Dans body :

```
<a href='#' onClick="decompte();return false">
    Compte 2 secondes
</a>
```

- On peut aussi utiliser setInterval() pour répéter la fonction de manière répétée, jusqu'à un appel de clearInterval() qui annule le setInterval

Objet Location

- Si l'URL est http://www.monsite.com/index.html?param1=toto

- *location.href* → http://www.monsite.com/index.html?param1=toto
- *location.host* → www.monsite.com
- *location.hostname* → www.monsite.com
- *location.pathname* → /index.html?param1=toto
- *location.search* → ?param1=toto

- Pour faire une redirection, il suffit de forcer les URL en donnant une valeur à window.location : `window.location = "http://www.google.com/"`

Récupération des paramètres URL

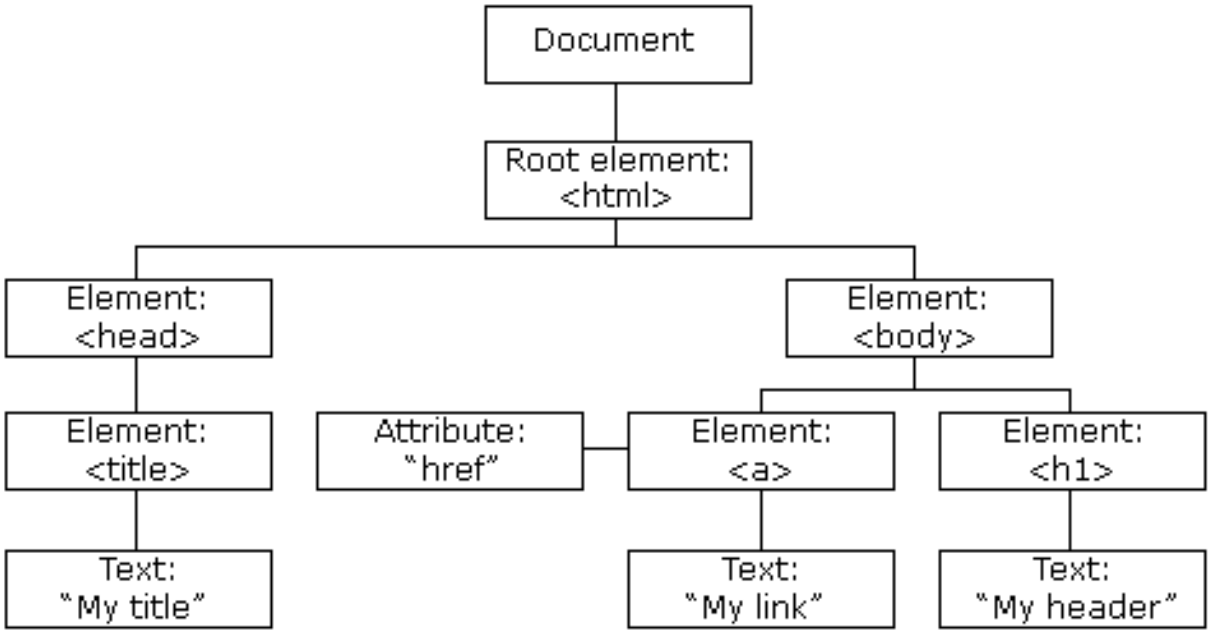
```
function getUrlParameter(name) {
    var searchString = location.search.substring(1).split('&');
    for (var i = 0; i < searchString.length; i++) {
        var parameter = searchString[i].split('=');
        if(name == parameter[0])    return parameter[1];
    }
    return false;
}

//utilisation : getUrlParameter("param")
```

Chapitre 3
Manipulation du DOM

Objets du DOM

- Le Document Object Model (DOM) est un standard du W3C pour hierarchiser le contenu d'un document XML ou HTML

- Comme pour le XML, une page HTML est constituée d'une série de nœuds ayant valeurs et attributs et des liens entre eux



Accès aux noeuds

- On peut faire référence à des noeuds de plusieurs façons :
- Par sa balise : document.getElementsByTagName("balise")
- Par son nom : document.getElementsByName("nom”)
- Par son id : document.getElementById("id")
- Par sa classe : document.getElementsByClassName("nom”)
- Par sélecteur CSS (un nœud) : document.querySelector("selecteur”)
- Par sélecteur CSS (plusieurs noeuds) : document.querySelectorAll("selecteur”)

- Attention : pour pouvoir accéder à un nœud, il faut que le code javascript soit exécuté APRES le chargement de la page. Pour ce faire, on appelle une fonction depuis la balise BODY comme suit : <BODY onLoad="init(); " > et on définit la fonction correspondante dans notre script

HTML collections

- Lorsqu'on utilise les fonctions getElementsByTagName ou getElementsByName, ou getElementsByClassName, ou querySelectorAll, javascript renvoie une HTMLCollection ou un NodeList, c'est à dire une liste de noeuds HTML

- Pour accéder au premier noeud, on peut utiliser : la méthode item(numero) ou bien utiliser les [crochets] comme les tableaux. Exemples :

```
getElementsByTagName('p').item(0); //retourne le 1er noeud
getElementsByTagName('p')[0] //idem
```

- Pour récupérer le nombre d'éléments de cette collection, on utilise la propriété length. Exemple :

```
getElementsByTagName('p').length;
```

Propriétés des noeuds

- noeud.nodeType → type de nœud

  1 : Élement **3 : Texte** **9 : Document**
  2 : Attribut **8 : Commentaire**

- noeud.nodeName → nom du nœud
  - Retourne la valeur si nœud est un attribut ou un élément.
  - Retourne #text oou #document le cas échéant.
  - Lecture seulement.

- noeud.nodeValue → valeur du nœud
  - undefined si nœud est un élément.
  - Utile pour les textes et attributs.
  - Pour les nœuds textes, nodeValue est l'équivalent de data.
  - Ré-écriture possible si le nœud existe déjà : nœud.nodeValue=valeur;

Propriétés des noeuds

- noeud.attributes → liste des attributs
  - Cette méthode récupère un nœud de type attribut. Elle est déconseillée car très incompatible ("a bloody mess") : http://www.quirksmode.org/dom/w3c_core.html#attributes
  - Il est préférable d'utiliser getAttribute() ou setAttribute() (voir ci-après)
  - attributes['id'] : attribut 'id' (cette méthode est presque compatible)
  - attributes.id : attribut 'id' (ne fonctionne pas avec class sur IE par exemple)
  - attributes[0] : premier attribut (non reconnu avant IE8)

- noeud.innerHTML → le contenu du nœud
  - Ecriture possible : nœud.innerHTML=valeur

Modifier le contenu d'un noeud

- Au niveau du contenu html
  - utiliser innerHTML en écriture :

nœud.innerHTML= valeur;

Modifier les attributs d'un noeud

- Au niveau des attributs
  - noeud.setAttribute(nom,valeur); → ajoute un attribut
  - noeud.getAttribute(nom); → lit un attribut
  - noeud.removeAttribute(nom); → supprime un attribute

- Ne pas utiliser ces fonctions pour l'attribut "class", "style" et les attributs d'évènements (onClick, onLoad, etc.)

Insertion d'id, class et style inline

- Ajout de l'attribut id :

  noeud.id="header"; → applique l'id "header"

- Ajout de l'attribut class :

  noeud.className="bt"; → applique la classe "bt"

- Ajout de l'attribut style :

  var style_complet ="border:5px solid black;background-color:red";

  noeud.style.cssText=style_complet; → applique le style complet

- Ajout d'un propriété CSS particulière :

  noeud.style.backgroundColor="blue"; → applique un fond bleu

Lecture d'id, class, et style inline

- Lecture de l’attribut id :

```
alert(noeud.id)
```

- Lecture de l’attribut classe :

```
alert(noeud.className)
```

- Lecture de l’attribut style :

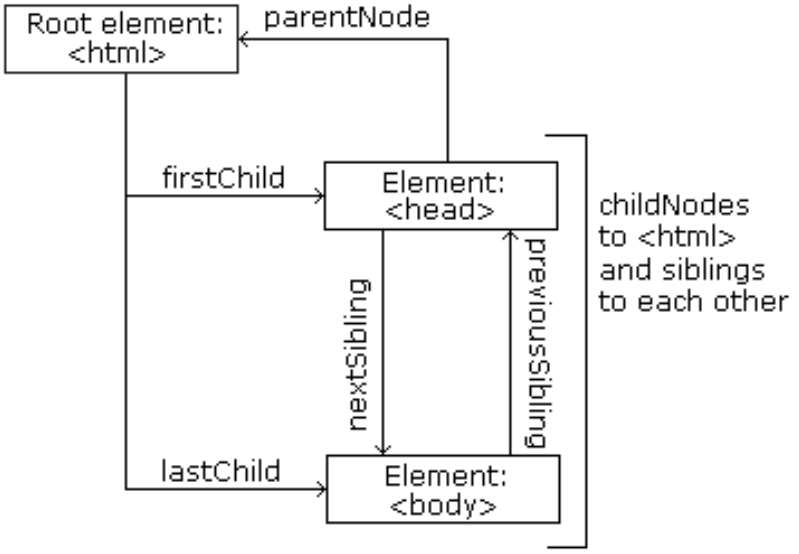
```
alert(noeud.style.cssText)
```

- Lecture d'une propriété CSS particulière :

```
alert(noeud.style.backgroundColor)
```

Se déplacer dans le DOM

- noeud.childNodes → les sous noeuds du noeud
- noeud.childNodes.length → nombre de sous noeuds du noeud
- noeud.childNodes[0] → premier sous noeud du noeud
- noeud.parentNode → nom du parent
- noeud.firstChild → premier sous noeud
- noeud.lastChild → dernier sous nœud
- nœud.previousSibling → nœud avant
- nœud.nextSibling → nœud après



Ajout de noeuds

- 3 étapes :
  - Création d'un noeud
  - Localisation de l'endroit où l'insérer
  - Insertion

- 1ère étape : création du nœud
  - Texte : document.createTextNode()

```
var texte = document.createTextNode("Nouveau texte");
```

  - Element : document.createElement() et noeud.setAttribute()

```
var lien = document.createElement('a');
lien.setAttribute('href', 'mapage.html');
lien.innerHTML = "bouton";
```

Ajout de noeuds

- 2nde étape : sélection de la destination (on pointe sur un nœud)
  - Utiliser les méthodes d'accès vues précédemment

```
var dest = document.getElementsByTagName("p").item(0);
```

- 3ème étape : insertion
  - Comme dernier enfant du nœud de destination :

```
dest.appendChild(lien);
```

  - Juste avant le premier enfant du nœud de destination :

```
dest.insertBefore(texte,dest.firstChild);
```

Suppression de noeuds

- 2 étapes :
  - Localisation du nœud où la suppression aura lieu
  - Suppression

- 1ère étape : localisation

```
var noeud = document.getElementsByTagName("p").item(0);
```

- 2nde étape : suppression

```
var disparu = noeud.removeChild(noeud.firstChild)
```

Chapitre 4
Événements et données

Évènements

- Un évènement est une action qui peut être détectée par Javascript
- Chaque élément de page web peut lancer un évènement, et une fonction Javascript est alors executée
- On reconnait un évènement car il commence par ‘on’ (onClick, onLoad, onMouseOver…)

- Références des Evènements :
  - http://www.w3schools.com/jsref/dom_obj_event.asp
  - http://www.quirksmode.org/dom/events/index.html

Ecouteurs d'évènements inline

- On peut ajouter un évènement à une balise, en lui disant quel code exécuter lors du déclenchement de l'évènement

- HTML :

```
<a onclick="clic_bouton(event)">mon bouton</a>
```

- Javascript :

```
function clic_bouton(e){
    alert("clic détecté");
}
```

Objet Event

- Le paramètre passé aux fonctions réagissants aux évènements est de type event. On s'assure qu'il soit récupéré pour tous les navigateurs ainsi :

```
function clic_bouton(e){
    if (!e) var e = window.event;
}
```

- On peut l'exploiter grâce à ses propriétés, notamment :
  - e.type
  - e.target (compatibles) ou e.srcElement (IE)
  - e.keyCode (compatibles) ou e.which (IE)
  - […] et toutes les propriétés propre à chaque type d'évènements

Ecouteurs d'évènements en JS

- On peut également éviter de placer un attribut sur la balise, et tout faire par javascript
- HTML

```
<body onLoad="init();">
    <a id='bt' href='#'>mon bouton</a>
</body>
```

- Javascript

```
function init(){
    var bouton = document.getElementById("bt");
    bouton.onclick=clic_bouton; //pas de ()
}

function clic_bouton(){
    alert('clic détecté');
}
```

addEventListener() et attachEvent()

- On peut aussi utiliser les fonctions avancées addEventListener() (compatibles) et attachEvent() (IE)

```
if(bouton.addEventListener){ //compatibles
    bouton.addEventListener('click',clic_bouton,false);
}else{
    bouton.attachEvent('onclick',clic_bouton); //IE
}

function clic_bouton(){
    alert('clic détecté');
}
```

Asynchronisme

- Un des paradigmes de programmation courante en JavaScript est le paradigme de la programmation asynchrone.
- Le principe (imagé) de la programmation asynchrone est de ne pas exécuter le code dans son ordre d'écriture. On parle d'opérations "non bloquantes".
- C'est particulièrement vrai dès lors que l'on travaille avec des événements :

```
bouton.onclick = function () {
    alert("clic");  //éxécuté lors d'un clic
}
console.log("A");   //executé en 1er
```

- On le voit aussi lorsque l'on fait de l'AJAX, car il y a un appel ver le serveur qui prend un certain temps avant de retourner une réponse au client.

Évènements window

- Evènements fenêtre
  - onload, onunload→ chargement, départ de la page
  - onresize → redimensionnement
  - onscroll→ scroll de la page

Évènements souris

- Evènements souris
  - onclick, ondblclick → clic, double clic
  - onmouseup, onmousedown → bouton levé, abaissé
  - onmouseover, onmouseout → entrée, sortie du pointeur
  - onmousemove → déplacement du pointeur

Évènements clavier

- Evènements clavier
  - onkeydown, onkeypress → touche enfoncée
  - onkeyup → touche relachée

Évènements formulaire

- Evènements formulaire
  - onfocus , onblur → gain, perte de focus
  - onselect → selection de texte d'un input ou textarea
  - onchange → modification d'un élément
  - onreset → réinitialisation du formulaire
  - onsubmit → envoi du formulaire

Détruire des écouteurs

- Si on a placé un écouteur via une balise :

```
noeud.removeAttribute("onclick");
```

- Si on a placé un écouteur via la méthode traditionnelle :

```
noeud.onclick=null;
```

- Si on a placé un écouteur via la méthode avancée :

```
if(bouton.removeEventListener){
    bouton.removeEventListener('click',clic_bouton,false); //compatibles
}else{
    bouton.detachEvent('onclick',clic_bouton); //IE
}
```

Chapitre 5
Gestion des formulaires

Rappels sur les formulaires

- <form> permet de délimiter un formulaire

- L'attribut method peut être égal à post ou get et permet de préciser la méthode d'envoi des données du formulaire. La valeur par défaut est get

- L'attribut action permet de spécifier l'URL de redirection lorsque le formulaire en envoyé. La valeur par défaut est l'URL actuelle (la page se recharge)

```
<form method="" post="" action="verif.php">
</form>
```

Rappels sur les <input>

- Les champs <input> prennent un attribut type pouvant prendre de multiples valeurs :
- <input type="text"> : champ de texte simple
- <input type="password"> : champ de mot de passe
- <input type="checkbox"> : champ de case à cocher
- <input type="radio"> : champ de bouton radio
- <input type="submit"> : bouton d'envoi
- <input type="file"> : champ d'envoi de fichiers
- <input type="reset"> : bouton de remise à zéro
- <input type="image"> : bouton d'envoi sous forme d'image
- <input type="hidden"> : champ invisible

- Chaque champ de formulaire doit avoir un attribut name pour transmettre sa valeur

```
<form method="post" action="verif.php">
    <input type="text" name="nom">
    <input type="text" name="email">
    <input type="submit">
</form>
```

Rappels sur les <select>

- <select> permet de définir une liste déroulante
- <select> contient des <option> qui correspondent à des lignes de cette liste
- Chaque option comprend une valeur entre ces balises correspondant à l'intitulé de l'option, ainsi qu'un attribut value correspondant à la valeur réellement envoyée par le formulaire
- On peut apposer un attribut selected sur une option, pour la définir en option par défaut

```
<select name="civilite">
        <option value="mlle">Mademoiselle</option>
        <option value="mme">Madame</option>
        <option value="m">Monsieur</option>
</select>
```

Rappels sur les <textarea>

- <textarea> permet de définir une zone de texte
- <textarea> peut prendre des attributs cols et rows permettant de dimensionner le champ


```
<textarea name="msg" cols="30" rows="10"></textarea>
```

Détection de l'envoi en JS

- On peut écouter l'évènement onsubmit sur <form> et appeler une fonction

HTML

```
<form method="post" action="verif.php"
onsubmit="verif()">
        <input type="text" name="nom">
        <input type="text" name="email">
        <input type="submit">
</form>
```

JS

```
function verif(){
   console.log("verif");
}
```

- Souvent on ne verra pas l'affichage en console car même si la fonction est appelée, la page verif.php est appelée tout de suite après. Pour bloquer l'envoi des données, on peut ajouter un return false à onsubmit :

```
<form method="post" action="verif.php" onsubmit="verif(); return false;">
```

- Mais dans ce cas, on intercepte toujours l'envoi des données vers verif.php. Or on veut seulement bloquer l'envoi du formulaire si verif() détecte des erreurs dans le formulaire les affiche. Comment procéder ?

Blocage conditionnel

- On va donc procéder ainsi :

HTML

```
<form method="post" action="verif.php" onsubmit="return verif()">
        <input type="text" name="nom">
        <input type="text" name="email">
        <input type="submit">
</form>
```

JS

```
function verif(){
    console.log("verif");
    return false; //bloque l'envoi
}
```

- Un return false dans verif() bloque l'envoi, mais aucun return ou un return true ne bloque pas l'envoi

- Lorsque nous aurons récupéré les champs et fait une détection d'erreurs, on pourra faire un return false en cas d'erreur, et un return true autrement

Récupération d'un champ

- Pour récupérer le champ nom dans verif(), on peut faire une sélection

HTML

```
<form method="post" action="verif.php" onsubmit="return verif()">[…]
```

JS

```
function verif(){
    let nom = document.querySelector("[name='nom']");
    console.log(nom);
    return false; //bloque l'envoi
}
```

- On peut également y accéder par son formulaire référencé dans document

HTML

```
<form method="post" action="verif.php" onsubmit="return verif()">[…]
```

JS

```
function verif(){
    let nom = document.forms[0].nom;
    console.log(nom);
    return false; //bloque l'envoi
}
```

Récupération d'un champ

- On peut aussi y accéder par le formulaire référencé dans l'évènement que l'on transmet à verif()

HTML

```
<form method="post" action="verif.php" onsubmit="return verif(event)">[…]
```

JS

```
function verif(e){
    console.log(e.target.nom);
    return false; //bloque l'envoi
}
```

- On peut aussi y accéder par le formulaire que l'on transmet à verif()

HTML

```
<form method="post" action="verif.php" onsubmit="return verif(this)">[…]
```

JS

```
function verif(form){
    console.log(form.nom);
    return false; //bloque l'envoi
}
```

Valeurs des champs

- Une fois qu'on obtient une référence à un champ, on peut accéder à sa valeur avec la propriété value

- On peut ainsi mettre en place une validation du formulaire, avec un booléen :

```
function verif(form) {
    let erreurs = false;              // aucune erreur par défaut
    if (form.nom.value == "") {       // test 1 : nom renseigné ?
        console.log("Merci de renseigner le champ nom.");
        erreurs = true;
    }
    if (form.email.value == "") {     // test 2 : e-mail renseigné ?
        console.log("Merci de renseigner le champ e-mail.");
        erreurs = true;
    }
    if (erreurs) {                    // si on détecte au moins une erreur
        return false;                 // on bloque l'envoi des données
    } else
        return true;                  // on ne bloque pas l'envoi
}
```

Validation du formulaire

- Amélioration de la validation, avec un tableau :

```
function verif(form) {
    let erreurs = [];           // aucune erreur par défaut

    if (form.nom.value == "") {   // test 1 : nom renseigné ?
        erreurs.push("Merci de renseigner le champ nom.");
    }
    if (form.email.value == "") { // test 2 : e-mail renseigné ?
        erreurs.push("Merci de renseigner le champ e-mail.");
    }

    if (erreurs.length) { // si on détecte au moins une erreur, on les affiche
        erreurs.forEach(function (erreur) {
            console.log(erreur);
        });
        return false;   // on bloque l'envoi des données
    } else
        return true;    // on ne bloque pas l'envoi
}
```

Validation du formulaire

- Affichez les erreurs en créant un conteneur de classe .erreurs, puis en faisant :

```
function verif(form) {
    let conteneurErreurs = document.querySelector(".erreurs");
    conteneurErreurs.innerHTML = "";
    let erreurs = [];           // aucune erreur par défaut
    if (form.nom.value == "") {   // test 1 : nom renseigné ?
        erreurs.push("Merci de renseigner le champ nom.");
    }
    if (form.email.value == "") { // test 2 : e-mail renseigné ?
        erreurs.push("Merci de renseigner le champ e-mail.");
    }

    if (erreurs.length) {// si on détecte au moins une erreur, on les affiche
        erreurs.forEach(function (erreur) {
            conteneurErreurs.innerHTML += `<p>${erreur}</p>`;
        });
        return false;   // on bloque l'envoi des données
    } else
        return true;    // on ne bloque pas l'envoi
}
```

Validation du formulaire

- On peut ajouter un test d'e-mail en utilisant les expressions régulières. On peut créer une fonction pratique :

```
function isEmail(mail) {
   var re = new RegExp('^[a-z0-9]+([_|\.|-]{1}[a-z0-9]+)*@[a-z0-
9]+([_|\.|-]{1}[a-z0-9]+)*[\.]{1}[a-z]{2,6}$', 'i');
   return (re.test(mail));
}
```

- Puis ajouter un test dans le fonction verif() :

```
// test 3 : e-mail est il au bon format ?
if (!isEmail(form.email.value)) {
   erreurs.push("Merci de renseigner un e-mail correct.");
}
```

Checkboxes et radios

- Lorsque l'on manipule des checkboxes ou des boutons radios, ces derniers seront regroupés par leurs noms. Côté JS donc, on récupère un tableau particulier : les RadioNodeLists. On peut les parcourir et faire référence à leurs propriétés checked pour savoir si leurs éléments sont actifs ou non

HTML

```
Disponibilités
<input type="checkbox" name="dispos" value="ouvres" checked> jours ouvrés
<input type="checkbox" name="dispos" value="samedis"> samedis
<input type="checkbox" name="dispos" value="dimanches">
```

JS

```
function verif(form){
   […]
   form.dispos.forEach(function(v,i){
        console.log(form.dispos[i].value+" : "+form.dispos[i].checked);
   });
   […]
}
```

Chapitre 6
jQuery

Frameworks Javascript

- Il existe des bibliothèques pour développer plus vite en Javascript
- Avantages :
  - Gain de temps
  - Dévelopement plus intuitif
  - Support multi-navigateurs
  - Nombreux plugins
  - Communauté

Installation de Jquery

- Téléchargement sur http://jquery.com/

- Placer le fichier dans un répertoire, créer une page web, et l'appeler depuis la page :

```
<script src="jquery.js" type="text/javascript" ></script>
        <script type="text/javascript">
        $(function(){
            alert('helloworld !');
        })
</script>
```

Sélections de noeuds : sélecteurs

- jQuery permet de sélectionner et d'agir sur un ou des nœuds directement. Exemple :

```
$("div").css("border","1px solid blue");
```

- On peut remplacer div par les sélecteurs suivants :
- #intro ou p#intro : les élements ou les paragraphes d'id intro
- .red ou p.red : les élements ou les paragraphes de classe red
- p strong : les éléments strong inclus dans des paragraphes
- a,strong : les éléments a et les éléments strong
- etc.

Sélections de noeuds : filtres

- On peut filtrer les listes de nœuds sélectionnées. Exemple :

```
$("div:nth(1)").css("border","1px solid blue");
```

- On peut remplacer :nth(1) par les filtres suivants :
- :nth-child (1) : les div 1ers fils de leur nœud parent
- :not (.red) : les div qui n'ont pas la classe red
- :empty : les div qui n'ont pas d'enfants
- :gt(1) : les div dont la position est supérieure à 1
- :lt(3) : les div dont la position est inférieure à 3
- :contains(3e) : les div qui contiennent le texte 3e
- [title=intro] : les div qui ont un attribut title = intro

Agir sur un noeud

- Styles :
  - Lecture : nœud.css("border")
  - Ecriture : nœud.css ("border","1px solid blue")
- Attributs :
  - Lecture : nœud.attr('title')
  - Ecriture : nœud.attr("title","second")
  - Suppression : nœud.removeAttr("title")
- Classes :
  - Ajout : nœud.addClass('red')
  - Suppression : nœud.removeClass('red')
  - Ajout / Suppression : nœud.toggleClass('red')
- Html :
  - Lecture : nœud.html()
  - Ecriture : nœud.html(…)
- Valeur :
  - Lecture : nœud.val()
  - Ecriture : nœud.val(…)

Faire des boucles

- On peut utiliser la fonction $.each() afin de boucler avec jquery :

```
$("li").each(function() {
    $(this).css('text-decoration','underline');
});
```

- Il n'est pour autant pas toujours nécessaire de l'utiliser :

```
$("li").css('text-decoration','underline');
```

- On peut aussi l'utiliser ainsi :

```
$.each( ["john","dave","rick","julian"] , function( index, value ) {
    console.log( index + ": " + value);
});

$( "li" ).each( function ( index ) {
    console.log( index + ": " + $( this ).text());
});
```

Créer, ajouter, supprimer un noeud

- Création :
  - `$("<div>nouveau div</div>")`

- Ajout :
  - Intérieur, au début : nœud.prependTo(nœud_destination);
  - Intérieur, à la fin : nœud.appendTo(nœud_destination);
  - Avant : nœud.insertBefore(nœud_destination);
  - Après : nœud.insertAfter(nœud_destination);

- Suppression :
  - vider : nœud.empty()
  - supprimer le nœud : nœud.remove()

Évènements

- On ajoute un évènement ainsi :

```
$('#bouton').click(function(){
        $(this).toggleClass("red");
})
```

- On peut remplacer click par :
  - focus / blur : gain / perte de focus
  - change : changement
  - dblclick : double clic
  - keydown / keypress / keyup
  - etc.

AJAX

- AJAX : Asynchronous JavaScript And XML
- Permet l'échange d'informations avec un serveur distant sans rechargement de page
- Asynchrone : le script n'attend pas la réponse pour continuer

AJAX

- Envoi de données en POST :

```
$.post("ajax.php",
       {mavariable:'mavaleur'},
       function done(data){
           alert(data);
       }
);
```

- Script serveur :

```
<?php echo 'Et voila : '.$_POST['mavariable'].' !'; ?>
```

Chapitre 7
Introduction aux APIs HTML5

API JavaScript

Contrôle audio
Contrôle vidéo
Site web hors ligne
Edition du contenu
Gestion des fichiers
Gestion du Glisser Déposer
Gestion de l'historique
Gestion des protocoles
Géolocalisation
Dessin 2D sur Canvas
Stockage web
Web Workers
Web Sockets
Messagerie inter-documents
Etc.

API Vidéo Javascript

- En javascript, on peut pointer sur une <video>, puis la contrôler ainsi :
  - video.volume=0.5; // volume
  - video.playbackRate=1; // vitesse
  - video.currentTime=0; // tête de lecture
  - video.play(); // lit la vidéo
  - video.pause(); // pause ou stop si on fait également currentTime=0

```
<video width="320" height="240" src="video.mp4"></video>
<br>
<a href="#" onclick="jouer()">Jouer</a>
<script>
    function jouer(){
        let video = document.querySelector("video");
        video.play();
    }
</script>
```

Jouer

API Audio Javascript

- En javascript, on peut pointer sur <audio>, puis la contrôler ainsi :
  - audio.volume=0.5; // volume
  - audio.playbackRate=1; // vitesse
  - audio.currentTime=0; // tête de lecture
  - audio.play(); // lit la vidéo
  - audio.pause(); // pause ou stop si on fait également currentTime=0

```
<audio controls src="audio.mp3"></audio>
<br>
<a href="#" onclick="jouer()">Jouer</a>
<script>
    function jouer(){
        let audio = document.querySelector("audio");
        audio.play();
    }
</script>
```

Jouer

Attributs data-

- Ils permettent de lier des infos à une objet et d'y accéder

- Exemple :

HTML :

```
<ul>
    <li data-nom="Dupont" data-ville="Lyon">John</li>
</ul>
```

JS :

```
var membre = document.getElementsByTagName("li")[0];
console.log(membre.dataset["nom"]);
console.log(membre.getAttribute("data-ville"));
```

Local Storage

- Permet de sauvegarder des données des couples clés / valeurs
- Les données sont stockées sur le navigateur
- Attention, pour IE, il est important que la page web soit accédée depuis un serveur et non depuis le système de fichiers
- Limite : 5M

- Test :
  if(localStorage){}

- Définir des valeurs :
  localStorage.setItem("cle","valeur");
  ou
  localStorage.cle="valeur";
  ou
  localStorage["cle"]="valeur";

- Récupérer des valeurs :
  localStorage.getItem("cle");
  ou
  localStorage.cle
  ou
  localStorage["cle"]

- Supprimer des valeurs :
  localStorage.removeItem("cle");

- Supprimer toutes les valeurs
  localStorage.clear()

Local Storage

- Exemple :

JS

```
window.onload=function(){
    if(localStorage['nom']){
        console.log("Bonjour "+localStorage['nom']);
    }
    if (localStorage) {
        var sauvegarder = document.getElementById('sauvegarder');
        sauvegarder.onclick = sauvegarde;
    }else{
        document.write("stockage web non supporté");
    }
}
function sauvegarde() {
    var nom = document.getElementById('nom');
    localStorage['nom'] = nom.value;
}
```

HTML

```
<input type="text" id="nom" placeholder="nom">
<a href="#" id="sauvegarder">Sauvegarder</a>
```

Session Storage

- Fonctionne de la même manière que localStorage, mais les données sont supprimées à la fermeture du navigateur

- Test :
  if(sessionStorage){}

- Définir des valeurs :
  sessionStorage.setItem("cle","valeur");
  ou
  sessionStorage.cle="valeur";
  ou
  sessionStorage["cle"]="valeur";

- Récupérer des valeurs :
  sessionStorage.getItem("cle");
  ou
  sessionStorage.cle
  ou
  sessionStorage["cle"]

- Supprimer des valeurs :
  sessionStorage.removeItem("cle");

- Supprimer la session
  sessionStorage.clear()

JSON.stringify()

- Pour transférer des objets, on doit avoir recours à JSON.stringify() et JSON.parse() car on ne peut transmettre que des chaines

Ecriture :

```
localStorage['obj'] = JSON.stringify({cle1:valeur1,cle2:valeur2});
```

Lecture :

```
var obj = JSON.parse(localStorage['obj']);
console.log(obj.cle1);
```

<canvas>

- <canvas> permettant d'importer et de créer toute sorte d'éléments visuels

- Référence : https://www.w3schools.com/tags/ref_canvas.asp

- Par exemple, dessinons un rectangle :

HTML

```
<canvas id="canvas" width="500" height="500"></canvas>
<button onclick="draw()">Dessiner</button>
```

JS

```
function draw() {
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext("2d");
    context.fillRect(50, 25, 150, 100);
}
```

Rectangles

- context.rect(x,y,largeur,hauteur) : rectangle sans fond / contour
- context.fillRect(x,y,largeur,hauteur) : rectangle avec fond
- context.strokeRect(x,y,largeur,hauteur) : rectangle avec contour
- context.clearRect(x,y,largeur,hauteur) : efface dans la zone donnée

Couleurs

- context.fillStyle="#DAEFE5" : permet de colorier le fond des prochaines formes
- context.strokeStyle="#DAEFE5" : permet de colorier leur contours
- les couleurs utilisables :
  - hexadecimal : "#000000"
  - rvb : "rgb(0,0,0)";
  - rvb transparente : "rgb(0,0,0,0.5)";
  - tsl : "hsl(196,50%,100%)";
  - tsl transparente : "hsla(196,50%,100%,0.5)";

Tracés personnalisés

- context.beginPath()
  débute une forme

- context.lineWidth=20
  définit la largeur de la ligne

- context.moveTo(x,y)
  déplace le point de contact

- context.lineTo(x,y)
  pour une ligne vers un prochain point

- context.stroke()
  trace l'ensemble des lignes

- context.closePath()
  termine une forme

```
context.beginPath();
context.moveTo(0,0);
context.lineTo(100,0);
context.lineTo(100,100);
context.lineTo(0,0);
context.strokeStyle = "#000";
context.stroke();
context.closePath();
```

Textes

- context.font = "bold 12px sans-serif" : définit la police
- context.fillText("abc",x,y) : écrit abc à la position x y

```
context.font="bold 12px sans-serif";
context.fillText("x",248,43);
context.fillText("y",37,255);
```

Images

- context.drawImage(image, x, y) ou
  context.drawImage(image, x, y, largeur, hauteur) : ajoute une image

```
//image à partir d'un fichier image.jpg
var img = new Image();
img.src = "image.jpg";
img.onload = function() {
    context.drawImage(img,0,0);
}
```

API Géolocalisation

- Permet de récupérer les coordonnées géographiques des utilisateurs l'acceptant
- Nous allons ici récupérer des coordonnées, mais nous pouvons les combiner avec des APIs comme google maps ou leaflet pour afficher une carte

- On peut effectuer la commande suivante :

```
if(navigator.geolocation){
    navigator.geolocation.getCurrentPosition(showMap);
}
function showMap(position){
    //récupération des variables
    console.log(position);
}
```

- Penser à executer le script via un serveur pour des raisons de sécurité
- On peut remplacer getCurrentPosition() par watchPosition() pour récupérer la position de manière continue. On peut arrêter le suivi par clearWatch()

Récupération des variables

```
function showMap(position) {
    console.log(position.coords.latitude);
    console.log(position.coords.longitude);
}
```

propriété	type	Notes
coords.latitude	Double	Degrés décimaux
coords.longitude	Double	Degrés décimaux
coords.altitude	Double / null	Mètres
coords.accuracy	Double	Mètres
coords.altitudeAccuracy	Double / null	Mètres
coords.heading	Double / null	Dégrés depuis le vrai nord (sens montre)
coords.speed	Double / null	Mètres/seconde
timestamp	DOMTimeStamp	Comme une objet Date()

Webliographie

http://www.w3schools.com/js/

http://www.w3schools.com/jsref

http://fr.selfhtml.org/javascript/

http://jacques-guizol.developpez.com/javascript/Objets/Objets.php

http://www.commentcamarche.net/contents/javascript/

http://jpvincent.developpez.com/tutoriels/javascript/javascript-oriente-objet-syntaxe-base-classes-js-intention-developpeurs-php/

http://www.javascriptkit.com/javatutors/proto4.shtml

http://www.w3schools.com/htmldom/dom_nodetree.asp

http://www.nanoum.net/blog/9_setAttribute.html

http://www.toutjavascript.com/savoir/savoir06_1.php3

http://ppk.developpez.com/tutoriels/javascript/gestion-cookies-javascript/

http://www.quirksmode.org/dom/intro.html

HUMAN**booster**

VOTRE SOLUTION COMPETENCE

Tél. 04 73 24 93 11 – contact@humanbooster.com

www.humanbooster.com

Clermont-Ferrand Montpellier Lyon