

LE WEB AVEC UN TRUC EN PLUS EN 2.0

**ORSYS**  
formation

JavaScript, HTML dynamique



PRÉSENTATION LOGISTIQUE

**ORSYS & VOUS**



## Bienvenue chez Orsys

### Présentation



- Indépendant
  - Prestataire d'Orsys
    - Depuis 2011
- Animateur
  - Monsieur DESORBAIX  
Alexandre
- Développeur
  - software, client lourd ,serveur
    - » c, c++, c#, java, ...
  - web, client léger
    - » html, php, sql, js, css,...
- blogger sous Wordpress



Bienvenue chez Orsys  
logistique



- Nombre de jours :
  - 4 jours
- Horaires
  - Début : 9h
  - Fin : 17h30 approx.
- Pauses
  - 10h30 approx. Matin
  - 12h30 approx. Midi
  - 15h30 approx. Apres-midi



## Objectifs pédagogiques

- Découvrir les concepts avancés JavaScript ES5
- Mettre en pratique la Programmation Orientée Objet
- Maîtriser l'environnement de débogage
- Mettre en œuvre le framework JavaScript jQuery
- Manipuler les API JavaScript HTML5
- Appréhender la notion de JavaScript côté serveur avec Node.js
- Appréhender le contexte d'une application JS d'entreprise interne



## **CONFIGURATION DU POSTE**



## Windows Apache Mysql Php

WampServer	WAMP	XAMPP
<ul style="list-style-type: none"><li>Gratuit, pour le dev rapide à mettre en œuvre</li><li>Mdp root mysql: «»</li></ul> 		<ul style="list-style-type: none"><li>Gratuit, pour le dev rapide à mettre en œuvre</li><li>Multiplateformes (linux osx win)</li><li>Mdp root mysql: «»</li></ul> 

Voir article : <https://www.blogpipers.com/2015/06/lamp-stack-xampp-vs-wamp-vs-mamp/>

## Windows Apache Mysql Php

WampServer 

WAMP

XAMPP

 XAMPP

- Gratuit, pour le dev rapide à mettre en œuvre
- Gratuit, pour le dev rapide à mettre en œuvre

- Seule une version builder pourrait avoir un sens à être servie avec apache

- La couche php ne sera pas intéressante pour nous

- Pour le développement react nécessite vraiment l'outil de service pour le dev

- Sans Node js :

- Complexité de mise en œuvre
    - Dev
    - test

Voir article : <https://www.blogpipers.com/2015/06/lamp-stack-xampp-vs-wamp-vs-mamp/>

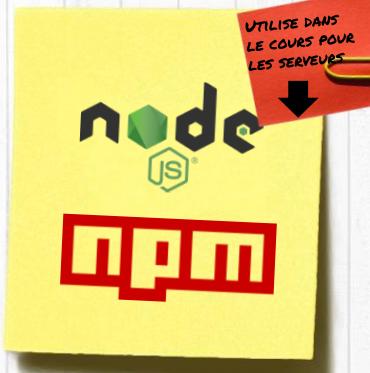
## Choix d'un serveur

- Servir du html en production :
  - Ils
  - Apache
  - Nginx, node, ...
- Dynamisation
  - Coté client js
    - Iis, apache, node, ...
  - Coté serveur js (code isomorphique)
    - Node
- Scafolder
  - create-react-app
    - Node.js
  - generate-react-cli
    - Node.js, ....

- ROLES
- SERVIR LES FICHIER
  - GÉRER LE JS COTÉ SERVEUR \*OPTION
  - OUTILS DE DEV

## Node.js & npm

- Node
  - Exécution de code JS coté serveur
    - Application
    - Scripting, ...
  - Lot d'applications existant
  - En ligne de commande
- Npm
  - Téléchargement de modules js
  - Gestion des dépendances



# npm

- Commandes npm

- Initialisation d'un répertoire de projet
  - Créer un fichier minimal package js
    - » **npm init -y**
      - Pré-configure node avec le répertoire...  
(si .git présent)
      - -y : initialisation « quiet »
- Installation des paquets
  - » **npm install -g packetname** packet2@version
    - -g : installation pour l'utilisateur courant,
    - sans -g installation pour le projet, dans le répertoire node\_modules
    - --save permet l'ajout du paquet aux dépendances du projet
- Npm start vs npm run monscript
  - Dans package.json, section scripts
  - Script name spécifiques nécessitant pas 'run' ex npm test, npm install, ...

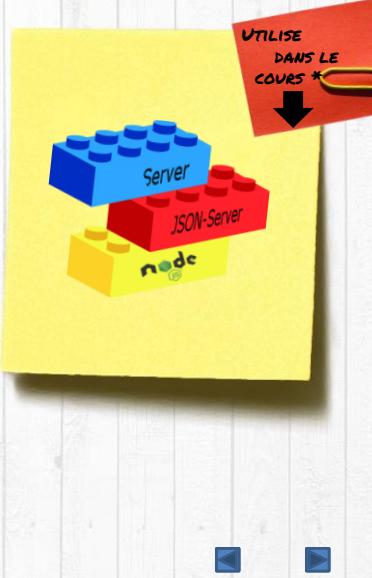


Il existe deux types de dépendances :  
devDependencies  
dependencies



## Serveur fake REST

- Json-server
  - Ecrit en node.js
  - Fake REST API
  - Mockeur de services rest
  - Stockage json
    - Création de endpoint en fonction des ressources du fichier
    - Ecriture direct sur le fichier json
  - <https://github.com/typicode/json-server>
    - npm instal -g json-server



UTILISE  
DANS LE  
COUER \*

\*ou wamp

**JSON-server tuto :** <https://codingthesmartway.com/create-a-rest-api-with-json-server/>

## Editeurs avancés pour le html/css/php/mysql/...

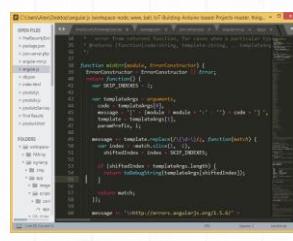
### VS Code

- Gratuit, multi langues, plugins & snippets, liaison debugger
- support GIT



### Sublime Text 3

- Gratuit, multi langues, plugins & snippets, liaison debugger



## Serveur GIT

- Plateforme de disposition de code versionné
  - Repository
  - Gestion des commit distant
  - Travail d'équipe
- Plateformes gratuite
  - Github, GitLab
    - Hébergé ou officiel
    - 1 repo / account
  - Altasian BitBuckets (\*Sélectionné pour le cours)
    - Multi-repository
    - Free (max 1Go/repo) ou payed account
    - Officiel uniquement



## Git - bitbuckets

- Création d'un compte gratuit
  - <https://bitbucket.org/account/signup/>
  - <https://bitbucket.org/account/signin/>
- Création 1ere repository
  - Edit, commit, approuve du readme.md
  - Clone local de la repository
- Ouverture du répertoire sous Visual Studio Code
  - Edition du readme.md sou vscode
  - 1<sup>er</sup> commit & push
- Approuve commit



## RAPPELS JAVASCRIPT : ES5, DOM, ÉVÉNEMENT



## Sommaire

- ❑ Les méthodes de création dynamique d'objets.
- ❑ Phases des événements capture, capturing, bubble.
- ❑ La gestion des événements. Les objets de type Event.
- ❑ Structure de données XML et JSON
- ❑ Les méthodes avancées en ES5.



## XML

- XML → eXstensible Markup Language
  - Version 1.0 & 1.1(pas utilisé)
  - XML est un langage structuré à balises
    - Data → Elément ( contenu de balise )
    - Meta-data → attributs
  - Il possède des règles syntaxique minimum
    - well-formdness rules
  - Il possède des mécanismes de validation natif dtd / xsd
  - Il est peut être parsé en document DOM
    - Fonctions DOM
      - getElement, append, remove, createElement, ...



## Syntaxe XML

Représentation	code XML - rss
<ul style="list-style-type: none"><li>• Processing instruction<ul style="list-style-type: none"><li>• facultatif</li></ul></li><li>• Un attribut<ul style="list-style-type: none"><li>• Dans une balise ouvrante</li><li>• nœud enfant de la balise</li></ul></li><li>• Une balise<ul style="list-style-type: none"><li>• ouverture &lt; Nombalise &gt;</li><li>• fermeture &lt;/Nombalise &gt;</li><li>• auto fermée &lt; Nombalise /&gt;</li><li>• du contenu enfant entre les deux</li></ul></li></ul>	<pre>&lt;?xml version="1.0" encoding="iso-8859-1" ?&gt; &lt;rss version="2.0"&gt;   &lt;channel&gt;     &lt;title&gt;Orsys&lt;/title&gt;     &lt;link&gt;http://www.orsys.fr&lt;/link&gt;     &lt;description/&gt;     &lt;item&gt;       &lt;title&gt;Formation XML&lt;/title&gt;       &lt;link&gt;http://orsys.fr/....html&lt;/link&gt;       &lt;description&gt;Concevoir       [...]XML...&lt;/description&gt;     &lt;/item&gt;     &lt;item/&gt;   &lt;/channel&gt; &lt;/rss&gt;</pre>

## « well formness/ Bienformité »

- Les règles de « bienformité » sont les règles minimal de syntaxe pour qu'un fichier XML puisse être fonctionnel
- Pour être valide le fichier doit avant le dtd/xsd être bien formé
- Il existe 8 règles en 1.0 et 6 en 1.1

Processing instruction XML en 1<sup>er</sup> caractère

Une seul balise racine

Pas de balise non fermé

Pas de balises croisées

Pas d'attribut sans valeur même null ( attr="" )

Le nom des balises commencent par une lettre ou '' (underscore)

Le nom des attributs commencent par une lettre ou un '' (underscore)

Les valeurs d'ID commencent par une lettre ou un '' (underscore) nécessite un dtd/xsd

## JSON

- JSON → Javascript Object Notation
  - Un format concurrent à XML
  - JSON est un langage structuré basé sur la syntaxe Javascript
    - Association clefs / valeur
    - Différents types de valeurs :
      - Object, array, number, string, null
  - JSON possède désormais une technique de validation
    - JSON Schema
      - intégrable en js jusqu'à la version 2019



# JSON

## Code JSON - geojson

```
{  
    "type": "FeatureCollection",  
    "features": [  
        {  
            "type": "Feature",  
            "geometry": {  
                "type": "Point",  
                "coordinates": [  
                    102.0, 0.5  
                ]  
            },  
            "properties": {  
                "prop0": "value0"  
            }  
        }, {"type": "Feature", [...]}  
    ]  
}
```

## Représentation

- Association

- Clef Toujours entre " "
- Valeur Plusieurs types

- Array
  - [ items ]

- Number

- Objet
  - { contenu }

- String
  - "Valeur de la chaîne"

- Séparateur de champs → ','

UN FLUX EST  
TOUJOURS  
CONTENU ENTRE  
{...}  
PRINCIPALES

## Sélecteur Css

- Eléments

```
p { font-size: 90%; margin: 0.5em; padding: 2px; }
```

- Classe

```
.important { color: red; }
```

- Identifiant

```
#menuGauche { margin: 0; padding: 0.5em; }
```

```
a:hover {  
border-top : 1px solid black;  
border-bottom: 1px solid black; }
```

- Pseudo-classe

23

**Élément :** un élément correspond à une balise (X)HTML telle qu'on la trouvera dans la structure du document, notée sans <> (h1, p, a). On peut ainsi définir des instructions CSS pour des éléments et façonnez un document de base.

**Classe :** lors de sa définition, une classe est identifiée par un point (.) précédent son nom (.maClasse). Vous pouvez choisir le nom que vous voulez, sans espaces. Elle pourra être utilisée sur différents éléments (X)HTML en leur ajoutant l'attribut class et la valeur de la classe entre guillemets sans le point (class="maClasse").

**Identifiant :** un identifiant fonctionne comme une classe. Il est référencé par un dièse (#) avant le nom. Un identifiant sert pour un élément (X)HTML unique en se basant sur l'attribut id des balises.

**Pseudo-classe :** les pseudo-classes permettent de mettre en forme des parties intégrantes du DOM. Il en existe peu :

- :active (élément actif),
- :focus (élément en focus),
- :hover (élément survolé),
- :link (non visité),
- :visited (élément visité) ,
- :first-child (premier élément fils),
- :lang (permet de spécifier une langue)

## Où écrire du JS

- La balise script

```
<script> console.log("hello");</script>
```

- Les fichiers js

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

- Dans les liens \*

```
<a href="JavaScript:alert('test');">Tester </a>
```

- Dans les events \*\*

```
<input type="button" onclick="alert('test');"/>
```

\* interdit par les règles d'accessibilité

\*\* mauvaise pratique à éviter

Dans la mesure où les navigateurs comprennent plusieurs langages, il faut leur préciser quand il doivent exécuter du JS. Pour cela, on place le code suivant : « `<script type="text/JavaScript">...code JS...</script>` ».

Remarque: « `language="JavaScript"` est déprécié depuis HTML4.01 soit 1999

## Conventions de nommage

- **Javascript, sensible à la casse**, utilise :
  - La **UpperCamelCase** (casse de chameau événement) pour les types / namespaces
    - Ex: RegExp ou String - Node ou Ajax
  - La **lowerCamelCase** pour les objets, variables, propriétés, fonctions, méthodes
    - Ex: window.clientHeight ou xhr.readyState - charAt(2) ou indexOf('?)
  - Tout en **majuscule**, chaque mot séparé par un « \_ » pour les **constants**
    - Ex: Node.TEXT\_NODE ou STR\_ERROR\_MSG
  - Tout en **minuscule** pour les **événements** et gestionnaires d'événement
    - Ex: onreadystatechange ou onclick
  - « \_ » **devant** le nom des propriétés **privées** (ex: \_myPvtProp)
    - Ex: var \_id = 0
  - un **verbe** en début de méthode et is/has/can pour celles **booléennes**
    - Ex: hasChild() ou isEmpty()

## règles syntaxique

- ; Point virgule signifie la fin d'une instruction
- . Point sert à accéder au méthode d'un objet
- () Parenthèses pour les appels et passages d'argument à une fonction
- [] Crochets pour accéder à l'élément d'un tableau
- {} Accolades pour délimiter un bloc d'instructions
- " Apostrophe \* & ""guillemets pour délimiter les zones de texte



\* conseillé

## Valuers particulières

- NaN → Not a Number , calcul impossible sur des valeur pas numérique ou division par 0
- null → Valeur non instacié
- *undefined* → methode ou propriété innexistante
- "" ou " → chaîne vide



# Variables

Une **variable** est un espace de stockage nommé. La **déclaration** de cet espace permet d'y **affecter** une valeur accessible par le nom de la variable.

Pour créer une variable on utilise le mot réservé "**var**", suivi du nom de la variable et/ou d'une valeur par défaut ( **new** pour construire un type de données )

- **var strMsg = 'message à afficher !';**
- **var strMsg = new String('message à afficher');**
- **var strMsg = new String;**  
**strMsg = 'message à afficher';**

**Remarque:** le mot réservé "var" n'est pas obligatoire, il permet seulement de limiter l'existence de la variable au bloc de code dans lequel elle a été créée.

Ceci évite les problèmes de collision avec d'autres variables du même nom créées dans d'autres blocs de code.

JAVASCRIPT HTML DYNAMIQUE

28

## PRECISIONS

- Pour ceux qui pratiquent des langages tels que le C, JS étant souvent interprété (cf. linker), ne séparez pas définitions, déclarations et init.
- On déclare une variable par : « var variable = valeur (typée); » ou « var variable = new Classe(); » ex: var intNum = 5; var strTmp = new String();  
**Remarque:** faites attention aux problèmes de linkage avec this et son utilisation combinée avec new, les méthodes (anonymes ou non), les gestionnaires d'événements...
- Pour les fonctions/méthodes on met le terme « fonction » suivi du nom de la fonction et de parenthèses contenant les paramètres à transmettre.  
cf: function fonction( param1, param2...){...code...}  
ex: function check( prmStrTxt){ var intPos=prmStrTxt.indexOf("?"");...}
- La zone entre parenthèses est réservée à la transmission de paramètres (vide si rien) (nb libre par rapport à la définition qui ne sert qu'à nommer les params). Ex: function validForm() { document.f.submit(); }
- La valeur de retour est renvoyée par la fonction « return(valeur); » simplifiée en « return valeur; » pour compatibilité avec les anciens codes.

## Types

- Types de données
  - numérique
    - **Number**
  - Etat booleen
    - **Boolean**
  - Objet
    - **Object**
  - Chaine de caractères
    - **String**
  - Tableaux
    - **Array**

### ATTENTION

LES VALEURS  
DÉCIMALES SONT  
SIGNIFIÉ PAR

• (POINT)  
ET NON PAR  
• (VIRGULE)

Preferé les types natif plutôt que les types objets

ex

```
var str='blabla' plutôt que var str=new String('blabla');  
var arr=[] plutôt que var arr= new Array();
```

# typeof

- Détection de type d'un variable

- typeof

```
console.log(typeof 42);
// expected output: "number"

console.log(typeof 'blubber');
// expected output: "string"

console.log(typeof true);
// expected output: "boolean"

console.log(typeof
declaredButUndefinedVariable);
// expected output: "undefined";
```



## Transtypage

- Conversion de type
  - Chaine vers numérique

```
Number.parseFloat(maStringVar);  
//conversion vers decimal  
Number.parseInt(maStringVar);  
//conversion vers entier
```

- Chaine vers Date

```
var date1 = new Date('December 17, 1995  
03:24:00');  
// Sun Dec 17 1995 03:24:00 GMT...  
  
var date2 = new Date('1995-12-  
17T03:24:00');  
// Sun Dec 17 1995 03:24:00 GMT..
```



## Opérateurs Logiques & Arithmétiques

- Opérateurs arithmétiques

$+, - \rightarrow$  addition, soustraction

$++, -- \rightarrow$  incrémentation ou décrémentation

$*, / \rightarrow$  multiplication, division

**mod**  $\rightarrow$  modulo (reste de division euclidienne)

- Opérateurs logiques

**&&**, **||**  $\rightarrow$  et, ou

**<, >, <=, >=**  $\rightarrow$  sup, inf, inf ou égal, sup ou égal

**!**  $\rightarrow$  NON booleen

**==, !=**  $\rightarrow$  strictement égal, différent

**====, !===**  $\rightarrow$  égal en valeur et en type,  
différent en valeur et en type



## Les Boucles

- Tant que

- Le nombre de tours est inconnu
- La condition est vérifiée au début de chaque tours
- Exécution de contenu pas obligatoire

```
while(maValeur != 10 )  
{  
    maValeur=instruction();  
}
```



## Les Boucles

- Faire ... tant que
  - Le nombre de tours est inconnu
  - La condition est vérifiée à la fin de chaque tour
  - Exécution de contenu obligatoire au moins un fois

```
do{  
    maValeur=instruction();  
}while(maValeur != 10 );
```

**ATTENTION**  
UN ; (POINT VIRGULE)  
EST NÉCESSAIRE  
APRÈS LA FIN DE LA  
CONDITION DU  
WHILE

## Les Boucles

- Pour

- Le nombre de tours est connu
- un l'ittérateur est instancié au départ
- l'ittérateur est incrémenté a chaque tour
- La condition est vérifiée a la fin de chaque tours
- Exécution de contenu obligatoire

```
for(var i=0; i<10;i++)  
{  
    instruction();  
}
```



## Array

- Initialisation groupée :  

```
var prenoms = ['Alex', 'Pascal', 'Laurent'];
```
- Lire ou modifier par l'index :  

```
prenoms[0] = 'Nikola';
```
- Taille d'array  

```
prenom.length
```
- Sélectionner :  

```
slice( idxDbt[, idxFin])
```
- Parcours avec for in  

```
for( i in prenoms){...code...}
```



## Array

- Listage

```
prenoms.join('');
```

– (join() sep = « , ») et toString() (idem join() )

- Tri

```
prenoms.sort(); ou
```

```
numbers.sort( function(a,b){return a - b})
```

- Concaténation de tableaux

```
prenoms.concat( prenoms2);
```



# Array

- Supprimer des éléments :
  - `pop()`
    - le dernier et le renvoie,
  - `shift()`
    - le premier
- Ajouter des éléments :
  - `push(a,b,...)`
    - à la fin
  - `unshift(a,b,...)`
    - au début
  - `splice( idx, nbr, emt1, emt2...)`
    - à partir de l'index
- Inverser l'ordre : `reverse()`

Afficher le code : `toSource()` presque en notation JSON (pas sous MSIE6-)



## Les test if/else if / else

- si condition

**if** (condition){instruction;}

- sinon si condition

**else if** (condition){instruction;}

- sinon

**else** {instruction;}

- ternaire

(condition)? SiVrai: SiFaux;



## les test switch/ case / default

- selon cas

```
switch (variable)
{
    case value :    instruction;
                    break;
    case value :    instruction;
                    break;
    default: instruction; break;
}
```

\*le cas **default** est une bonne pratique

## with

- Simplification avec with quand on veut faire plusieurs opérations sur un objet, on peut utiliser with

```
with ( objTxt.style) {  
    border = '1px solid black';  
    fontSize = '12px';  
    fontWeight = 'bold';  
}
```



# Fonctions

- Déclaration de fonction

- Valeur de retour facultatif
- arguments d'entrée facultatif

```
function nomFonction(argument1, argument2)
{
    instruction(argument1);
    var uneValeur= argument1 + argument2;
    return uneValeur;
}
var nomFonction2= nomFonction;
```

- Appelle de fonction

- Affectation du retour et variable d'entrée facultatif

```
var z=nomFonction(123, uneValeur);
nomFonction2(123, uneValeur);
```



## Fonctions

- Les arguments d'entrées non déclarées

- un tableau des variables d'entrées auto déclarées
- si pas d'argument d'entrée déclarée lors de la déclaration de la fonction
- Mot clef : **arguments**

```
function nomFonction(){
    console.log(arguments[0]);
}
nomFonction(123);
```



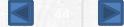
## Gestion des exceptions

En JS on peut aussi gérer les erreurs avec **try** (essaye), **catch** (attrape l'erreur), **finally** (finalement fait) et **throw** (renvoi l'erreur).

Les exceptions prédéfinies sont  
Eval/Range/Reference/Syntax/Type/URI/InternalError.

```
try
{
    ... code qu'on essaye d'exécuter
}
catch(e)
{
    ... code d'autres erreurs
} finally
{
    ... exécuté obligatoirement
}
```

JAVASCRIPT HTML DYNAMIQUE



## C'est l'histoire d'un mec...

- C'est l'histoire d'un mec, de 56 ans habitant à mullhouse prénomme jean-pierre



- Il acheté une machine à café, le type ...
  - Elle à un bouton marche / arrêt
  - Un bac d'eau à remplir d'un litre
  - Et du café en poudre à mettre dans le filtre à café

- Analysons la conception de l'objet



## C'est l'histoire d'un mec...

- Sans intérêt pour nous
- C'est l'histoire d'un mec, de 56 ans habitant à Mulhouse prénommé jean-pierre
- Identifions une class machine à café



- Des fonctions d'interaction public
  - Elle a une fonction marche / arrêt
  - Un champ définissant le volume d'eau du bac
  - Elle possède un champ de type café
- Des fonctions et champs qui lui sont propres
  - Température de l'eau
  - Activation de la résistance chauffante
  - Activation de la pompe
  - Eclairage du bouton lors du branchement
- Une cafetière c'est donc un objet!!!  
Et jean-pierre aussi



Heu l'autre!!



## C'est l'histoire d'un mec...

- Jean-pierre est pas adroit il met du café en poudre partout tous les matins
- Il acheté une machine à café,
  - oui encore, système café à dosettes cette fois
  - elle fait tout pareil ,mais plus pratique
  - - Elle à un bouton marche / arrêt
    - Un bac d'eau à remplir d'un litre
    - Et un objet filtre qui contient café est redéfini différemment
      - Il accepte désormais du café en dosette
- Alors une machine à dosette c'est une
- machine qui fais aussi du café.
- Comme l'autre!!!



On parle ici de l'extension d'une class  
qui reprend tout ce que fais un objet pour faire un nouvelle objet a partir du premier.

Il peut redéfinir des fonctions ou méthodes existante chez le parent et en apporter d'autres.

Ici notre cafetier posséde les memes fonctions : faire le café par exemple

Mais le fonctionnement n'est pas profilement le meme.

Le café cette fois , toujours présent mais un nouveau conteneur doit etre insérer dans la machine , ... plus le café directement

## Class machine à café

- Machine à café à dosettes
  - Ses champs privés
    - Ceux du parent étant eux aussi private il sont innaccessible
      - » Notion d'heritage
      - » Notion de protected
  - Des champs / fonctions public, protected
    - Permettant l'héritage aux futurs extensions
    - Redéfinition de la manière de faire le café
      - » Notions d'override
  - Des champs / fonction privés qui lui sont totalement personnel
  - Un constructeur
    - Appel du constructeur parent
      - » Notion de super()
  - Des fonctions interne protégé pour permettre aux futurs génération de pleinement hérité de leur ainés

```
CLASS COFFEMACHINPAD EXTENDS  
COFFEMACHINE{  
    PUBLIC PADHOLDER: PADHOLDER;  
  
    CONSTRUCTOR()  
    {  
        SUPER();  
    }  
  
    PUBLIC INSERTDOSET(PAD:COFFEPAD){...}  
  
    PUBLIC MAKECOFE(){  
        THIS.COFFEE= THIS.PADHOLDER.COFFE;  
        THIS.STARTHEATER(77);  
        THIS.STARTPUMP(255);  
    }  
};
```

## C'est l'histoire d'un mec...

- Jean-Pierre est devenu un expert du café
  - Il a essayé une 20aine de systèmes différents.
- Il à finit par déduire que
  - Faire du café c'est toujours :
    - Mettre du café/ eau
    - Démarrer
    - Collecter boisson chaude
  - Il a aussi identifié 2 sous modèles génériques pour faire du café
    - Les machines classiques
    - Les machines a expresso
  - abstraction de faire le café
    - Les machines implémentent les fonctions de faire du café commun et possède des champs communs
    - Il peuvent grâce à **implements** bénéficier de l'abstraction d'autres fonctionnalités, ex : mousseur à lait



On parle d'abstraction.

Dans l'abstraction on définit (ou uniquement le besoin de de déclarations), le contenu commun a implémenter chez l'enfant

L'abstraction est plus a voir comme la définition d'un *Object* générique plutôt *sous forme de besoins conceptuel*

Contrairement à **extends** , **implements** se limite à forcer l' éxistance de fonctions ou champs pour chaque implem.

## Notions d'objet

- En JavaScript, tout ceci donne :

Function.js

```
function Custom(props){  
    this.plane=function(){...};  
    function init(){  
        this.couleur=props.color;  
    }  
    init();  
}  
var myCutsom = new Custom({color:1});  
myCutsom .couleur = 'marron';  
myCutsom .plane();  
myCutsom.onCustomEvent= function() { ... }
```

JAVASCRIPT HTML DYNAMIQUE



# Notions d'objet

Il existe des mots clés réservés à la manipulation des objets.

## this

- this = objet courant, à savoir:
  - niveau global : objet window courant (frame)
  - fonction / méthode : l'objet associé à la méthode
  - événements : objet déclencheur.

## new et delete

- opérateur new pour créer (alloquer) un nouvel objet.
- chaîne non-allouée et null
- pour dés-allouer utiliser delete (à noter : JS ne fournit pas de gestionnaire de destructeur car il dispose d'un ramasse-miettes = « garbage collector »)

JAVASCRIPT HTML DYNAMIQUE



## this

- this est un mot réservé qui fait référence à l'objet courant, c'est à dire : au niveau global, l'objet window courant (frame), dans une fonction (méthode), l'objet associé à la méthode (peut-être window pour les fonctions globales), dans les événements c'est l'objet déclencheur.

## new et delete

- l'opérateur new est un opérateur spécial qui permet de créer (d'alloquer) un nouvel espace mémoire pour l'objet.
- une chaîne pour laquelle aucun espace mémoire n'est allouée est null
- pour dés-allouer la mémoire qui a été réservée, il faut utiliser l'opérateur delete. C'est important si on manipule de grandes quantités de données (ex: Array de grande taille).

## Notions d'objet

### Des objets ou des classes ?

- Si on peut étendre les types existants à l'aide de la méthode prototype, les langages de prototype ne permettent pas de créer des classes.
- On peut néanmoins simuler les classes et espaces de noms en utilisant des conteneurs comme suit : `var Ajax = {}... Ajax.Updater = { ... code ... }`
- Et simuler des classes en combinant conteneurs, la portée des variables et méthodes, l'usage de la méthode prototype...
- NB: utilisez des fonctions anonymes pas new Function sinon il risque d'y avoir des problèmes de linkeur.



## Notions d'objet

### Implémentation de classes

- les propriétés privées sont déclarées dans l'objet avec 'var' on ne peut y accéder que par des méthodes privées ou privilégiées.
- les méthodes privées déclarées inline dans le constructeur ou par : « `var functionName = function(){...}` » (même portée que précédent).
- les méthodes privilégiées déclarées par : « `this.methodName=function(){...}` » et invoquées depuis l'extérieur de l'objet.
- les propriétés publiques déclarées par « `this.variableName` » et lues et modifiées depuis l'extérieur de l'objet.
- **JavaScript ne supporte pas les variables et méthodes protégées !**



## Notions d'objet

### Implémentation de classes

- appelées depuis l'extérieur de l'objet.
  - les méthodes publiques sont définies par « `Classname.prototype.methodName = function(){...}` »
    - appelées depuis l'extérieur de l'objet.
  - les propriétés de prototype par : « `Classname.prototype.propName = « value »` »
  - les propriétés **statiques** définies et utilisées par « `Classname.propName` »
  - **JavaScript ne supporte pas les variables et méthodes protégées !**



# Notions d'objet

## Simuler l'héritage

- Une **classe peut hériter** par « `ChildClassName.prototype = new ParentClass();` »
- Il faut penser à **rétablissement immédiatement le constructeur**, par :  
« `ChildClassName.prototype.constructor = ChildClassName;` »
- On peut appeler les méthodes des ancêtres (pas leurs propriétés privées) que vous avez surchargées à l'aide de la méthode `Function.call()`, ex :  
« `Chat.prototype.dors = function(){ Mammifere.prototype.dors.call( this);...}` »
- Pour garder une trace du parent (afin de ne pas avoir à se souvenir de tous les parents / enfants) on peut créer une propriété privée « `parent` », ex :  
« `Chat.prototype.parent = Mammifere.prototype;... this.parent.dors.call(this);` »
- On peut simplifier la notation en créant une fonction par prototypage sur le type `Function` (ex: `heriteDe`) et en instanciant les objets comme des fonctions.

# Principaux objets

## Fonctions/Classes globales (native ou built-in)

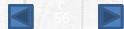
- built-in = intégrées nativement (ex: `Boolean`, `Number`, `String`, `undefined`...). Accessibles partout et sans propriétaire.

## Les chaînes de caractères (String)

Quelques propriétés et méthodes des chaînes de caractères:

- `String()` son constructeur, ex: « `var strTxt = new String();` »
- `indexOf` position d'une sous-chaîne ex: « `strTxt.indexOf('?)';` »
- `charAt()` le caractère situé à une position (index) ex:  
« `strTxt.charAt(3);` »
- `length` propriété indiquant la longueur de la chaîne

JAVASCRIPT HTML DYNAMIQUE



## Fonctions/Classes globales (native ou built-in)

- Certains objets, classes et fonctions sont built-in, c'est à dire intégrées nativement à JS. C'est le cas de `Boolean`, `Number`, `String`, `undefined`... Les fonctions built-in sont accessibles partout et n'ont pas de propriétaire.

- Les fonctions `parseInt` et `parseFloat` sont Built-in. « `parseInt` » renvoie un entier sur la base d'une chaîne (interprétée) et d'une base mathématique (ex: base 10 ou base 16). Ex: « `parseInt('152.25', 10);` » renvoie « 152 » et « `parseInt('0xA1');` » renvoie « 161 ».

Remarques: les chaînes commençant par « 0x » sont interprétées en base 16 par défaut, celles commençant par 0 comme de l'octal (d'où certains déboires parfois avec les dates), les autres en base 10.

# Principaux objets

## La classe statique Math

La classe Math est particulière et utile pour des calculs, ex:

- `Math.floor(valeur)` renvoie l'entier inférieur (pas la partie entière !)
- `Math.random()` renvoie un chiffre à virgule aléatoire entre 0 et 1

## Gestion du temps (Date)

Quelques informations, propriétés et méthodes pour bien gérer les dates:

- Rappels sur les dates (GMT, US vs FR, Julian, DateStamp...)
- `Date()` constructeur, ex: « `datTmp = new Date();` » (date affectée ici)
- les méthodes `set/getTime(tps en sec)` permettent de lire et de fixer l'heure
- la méthode `toGMTString()` convertit une heure en heure GMT (décalage)



# Principaux objets

## screen, window, navigator

- **screen** = écran de l'internaute (réso: width / height)
- **navigator** = navigateur de l'internaute (MSIE, MFF), (info de param)
- **window** = fenêtre courante (frame si frameset)

## document

- **document** = page actuelle (write, clientWidth/Height, arbre DOM)

## e-mails (« mailto » pas js mais assimilée)

- mailto commande HTML assimilée JS pour ouvrir le maileur paramètres : adresse, subject, body...
- tel, sms fournit par html5

## PRECISIONS

- L'objet screen correspond à l'écran de l'internaute, on peut par exemple récupérer screen.width, screen.height pour connaître sa résolution

- L'objet navigator correspond au navigateur de l'internaute (MSIE, MFF), on peut récupérer des informations de paramétrage afin de personnaliser (si besoin) le code. Ex:

« navigator.javaEnabled(); » permet de savoir s'il y a une JVM  
 « navigator.appName.indexOf("Netscape"); » pour ce type de navigateur

- L'objet window correspond à la fenêtre courante (frame si frameset), on peut par exemple récupérer l'URL (pratique pour la méthode GET),  
 ex: « window.location » et « window.location.search »  
 [window].alert('message'); pour afficher un message dans une dialogbox  
 [window].history[.go(-1)]; pour revenir 1 action en arrière  
 [window].close(); pour fermer la fenêtre courante  
 window.open(URL, nom, frame, params) (précis & sans espaces)

## document

- L'objet document correspond à la page chargée, on peut utiliser :  
 document.write('texte à écrire') pour écrire dans la page  
 document.body.clientWidth (ou Height) pour savoir la taille intérieure de page  
 document.forms[0], document.layers['lyr1'] pour accéder aux objets (DOM0)

- Par exemple, avec un script simple on peut procéder à des permutations d'images. « documents. images[0].src = ».

## e-mails (« mailto » pas js mais assimilée)

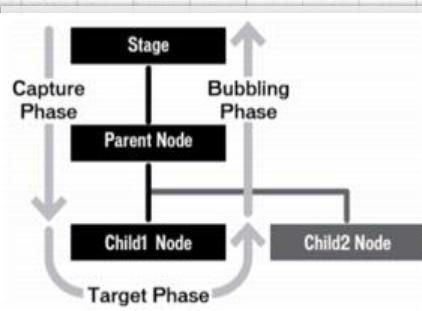
- La commande mailto est une commande HTML souvent assimilée à du JS, elle permet d'ouvrir l'outil d'envoi d'e-mails (MS OE, Mozilla Thunder Bird...) de l'internaute en lui transmettant des paramètres (sujet, adresse e-mail du destinataire, texte par défaut ...). On la met dans un lien.

- ex: « mailto:info@web.com?subject=Sujet%20du%20courrier&body=... »

## Organisation des événements

Un événement est un **stimulus propagé** lors d'une action **utilisateur** (click de la souris, mouseOver...) ou d'un traitement du **système** (chargement d'une image).

On gère sa réponse par la **programmation du comportement de l'interface** pour événement sur objet donné.



### 1 Capture

L'événement se propage depuis la racine du document jusqu'à la cible.

### 2 Ciblage

L'événement atteint la cible.

### • 3 Bouillonnement

- L'événement se propage dans le sens inverse.

Il est possible d'interrompre la propagation d'un événement donné par l'instruction `event.stopPropagation.`

JAVASCRIPT HTML DYNAMIQUE



## PRECISIONS

- Comme expliqué précédemment, les événements sont des stimuli. Les objets sont adaptés à répondre à certains stimuli et disposent donc de gestionnaires d'événements. Par exemple, quand on passe le curseur de la souris sur un lien, il reçoit le stimulus (MouseOver) « la souris est au dessus », il déclenche alors son gestionnaire d'événement onMouseOver (l'action à faire dans ce cas). Ainsi, en JS, pour indiquer ce que doit faire le gestionnaire d'événement, il suffit de mettre dans le code de l'objet : « `onMouseOver="...code JS..."` » (inutile de mettre « JavaScript: »).  
ex: `<input type="text" id="txtNom" name="txtNom" onClick="alert('test ');>`  
Remarque: ceci est un exemple de code obtrusive (non-compatible avec la séparation contenu/mef)

- notez que les messages d'événements envoyés ainsi que la manière dont ils sont envoyés dépendent du SE et du navigateur de l'internaute (qui offrent un environnement d'interaction à ces messages).

## Gestion des événements

Les événements dépendent du système d'exploitation et du navigateur utilisé pour leur capture. Mais **la nature de l'évènement dépend du modèle prévu dans l'interface un objet donné.**

### Événements notables

load

click

focus

keyDown / Up /Press

change

blur

unload

### Processus :

- Chargement du document
- Click dans le document
- Sélection d'un élément
- Utilisation du clavier
- Modification de la valeur d'un champ
- Sortie d'un élément
- Fermeture (déchargement) du document

## Objet Event

- **Event** est l'objet JavaScript décrivant les événements. C'est un objet de type Event qui se propage dans l'arbre du DOM.
- Propriétés :
  - **target** (MSIE srcElement) : la cible recherchée
  - **type** : le type de l'événement (focus, load, click)
  - **currentTarget** : nœud courant de l'arbre DOM
- Méthodes:
  - **stopPropagation** (MSIE cancelBubble)
  - **preventDefault** (return false)

## Gestionnaire d'événements

C'est une fonction JavaScript interceptant un évènement lors de sa propagation de l'arbre du dom.

Le gestionnaire (**écouteur ou listener**) associé à un objet par :

- La phase
- Le type

Pour gérer les gestionnaires d'événement sur un objet on utilise **addEventListener** et **removeEventListener**

```
function submitForm(event){  
    if(this.action == "") event.preventDefault();  
}  
Document.getElementById("formulaire").addEventListener("submit", submitForm, false);
```

## Gestion des événements

### Obtrusive vs unobtrusive JS

Les anciennes techniques prônent l'intégration de JS dans la page

HTML :

```
<script type="text/JavaScript" >...code...</script>  
<input onclick="...code..."
```

Aujourd'hui, dans **l'objectif XML de séparation du contenu**, de la forme et de l'interactivité, il faut intégrer le JS discrètement (includes) :

```
<script type="text/JavaScript" src="fichier_code.js"></script>  
document.forms[0].onsubmit = check; (DOM 0 → initEvents en  
fin de page)
```

JAVASCRIPT HTML DYNAMIQUE



## Gestion des événements

### Anciennes techniques (JavaScript obtrusive):

On crée une fonction permettant de réaliser l'opération désirée dans un fichier JS qu'on intègre à l'aide de la balise `<script>`

- Dans le HTML : `<script ... src="fichier.js">`
- Dans le JS : `function showMenu() { ... }`

On déclare la fonction à utiliser directement dans l'attribut événement de la balise HTML

- Dans le HTML : `<body onload="showMenu();"> ...`

JAVASCRIPT HTML DYNAMIQUE



## Gestion des événements

### Nouvelles techniques (**JavaScript unobtrusive**):

Aujourd'hui, pour respecter l'objectif sémantique, il faut utiliser du **JavaScript** discret. Plus aucun code **JavaScript** ne se trouve dans la page HTML.

On n'utilise pas les gestionnaires des balises mais l'affectation des gestionnaires d'événements en **JavaScript**

- Dans le JS : `document.body.onload = function() { showMenu();}`

JAVASCRIPT HTML DYNAMIQUE



#### Exemple de code JS discret :

```
function addEvent( emt, evt, fnc, bbl)
{
    if( emt.attachEvent) emt.attachEvent( 'on' + evt, fnc); //4 MSIE
    else if( emt.addEventListener) emt.addEventListener( evt, fnc, bbl); //4 ECMA ex: MFF
}

function initEvents()
{
    addEvent( document.getElementById('btn_cmd'), 'click', evalCmd, false);
}

function evalCmd()
{
...
}

addEvent( window, 'load', initEvents, false);
```

## Gestion des événements

### Bonne pratique:

- Dans le **JavaScript**, pour MSIE6-, on utilise **attachEvent** :  
`document.body.attachEvent('onload', showMenu);`
- Dans le **JavaScript**, pour les autres navigateurs, on utilise  
**addEventListener** :  
`document.body.addEventListener('load', showMenu, false);`

**Remarque :** il faut s'assurer que les objets HTML sont bien chargés avant de leur affecter des événements. Pour cela on ne lance les initialisations d'événements qu'après chargement de la page (événement load du body cf code ci-dessous).

**Remarque :** on détruit les écouteurs sur événement avec **detachEvent** et **removeEventListener**

## Javascript es5

- Les fonctions auto-exécutées.

Self-executed.js

```
(function(arg1){...})( "value");
```

↳ Limite la portée et l'accès post exécution



## Conclusion

- ✓ Les méthodes de création dynamique d'objets.
- ✓ Phases des événements capture, capturing, Bubble.
- ✓ La gestion des événements. Les objets de type Event.
- ✓ Structure de données XML et JSON
- ✓ Les méthodes avancées en ES5.



# FRAMEWORK JQUERY



## Sommaire

- ❑ Fonctionnement et intégration.
- ❑ Gestion et délégation des événements.
- ❑ jQuery pour des échanges Ajax, gestion des formulaires.



## jQuery

- Jquery développé par **john Resig**
  - Maintenu par jQuery team
- Permet d'écrire moins de code pour plus d'actions javascript
  - Permet une meilleur interaction
    - Ergonomie
    - Dynamisation
  - Moins de code = moins de maintient



- jQuery fournit un ensemble d'actions.

- Manipulation DOM
  - Selection , ajout, suppression, ...
- Gestion des events
  - Async avec les callbacks
- Animations & style css
  - Slide, up down, ...
- Interactions serveur
  - XHR simplifié



## Intégration

- Pour intégrer jQuery dans une appli :
  - Il suffit d'importer le script

```
<script src="//code.jquery.com/jquery.js"></script>
```

- Soit par CDN
  - **C**ontent **D**elivery **N**etwork
- Soit par son propre serveur



# JQuery

- jQuery est un framework Javascript
  - jQuery qui peut être raccourcis par \$ (le plus souvent)
  - Cette fonction fournit :
    - Une sélection de nœud simplifiée par sélecteur css

```
var uneDiv = $('div');
```
    - Une série de fonctions pour manager plus facilement différentes actions

```
uneDiv.slideUp(); / uneDiv.slideDown();
```

```
$.ajax({ url: "demo.html",
    success: function(result){
        $("#div1").html(result);
    }
});
```



## noconflict

- jQuery s'approprie le dollar \$ comme raccourcis

- En cas d'usage de ce raccourcis \$ par un autre framework,fonctions, ...

`jQuery.noConflict()`

- jQuery et la sélection

`jQuery('selecteur css') ➔ $('selecteur css');`



## filtres

- La sélection \$ renvoie une liste de nœuds
  - Limiter ou affiner la sélection grâce aux filtres.
  - Les filtres définissent soit des positions des comparaisons, des parcours, ...

selectionJQ.js

```
$('ul#liste').children().each(...);
```

- **FILTRES :**
- .addBack()
- .children()
- .contents()
- .each()
- .end()
- .eq()
- .filter()
- .find()
- .first()
- .has()
- .is()
- .last(), ...

## Gestion des events

- La fonction  
.bind(event[,eventData],handler)

```
$('.uneClasse').bind('click',function(event){  
})
```

- Event : nom de l'event
  - » Ici click
- Handler : fonction à exécuter lors de cette event

- *IL EXISTE DES EVENTS PRÉDÉFINIS :*

- .RESIZE(),  
.SCROLL(),  
.READY(),  
.FOCUS(),  
.SELECT(),  
.CLICK(),  
...*

## Les effets

- jQuery propose des animation sur la représentation html / CSS

- Du fading :

- \$(`#node`).fadeTo(), ou fadeIn(), fadeOut()

- Des toggle

- \$(`#node`).toggle(), ou .hide(), .show()

- Des sliding

- \$(`#node`).slideDown(), ou slideUp, slideToggle

- EXEMPLE

```
jQuery(document)
  .ready(function ($) {
    $('#toggable').
      click(
        function (e) {
          $('# toggable')
            .slideToggle();
        });
  });

```

# **CHEAT SHEET**



## \$.ajax

- jQuery fournit une fonction AJAX
  - » Asynchronous Javascript And Xml
  - La méthode ajax requiert un objet de config avec les paramètres suivants :
    - url : une chaîne url de destination de l'appel http
    - success : une fonction de callback en cas de succès
    - error : une fonction en cas d'échec \*facultatif

```
$.ajax({url: "demo.html",
        success: function(result){
            $("#div1").html(result);
        },
    });

```



## jQuery

- jQuery propose de gérer les events.
  - Comme fonction du nœud jQuery
    - Reçois une callback d'exécution pour l'event
      - \$('button#btn1').click(function (evt){...})
      - \$('#myForm').submit(function (evt){...})
      - , ...
    - Html possède des actions par défaut lié a certains event
      - Ex : *onsubmit* → aura pour action par défaut de faire l'appel http sur la *target* définit dans la balise *form*
  - Il est important de prévenir et stopper l'exécution de cette action pour laisser le contrôle a la callback de l'event:  
**evt.preventDefault()**



## Conclusion

- ✓ Fonctionnement et intégration.
- ❑ Gestion et délégation des événements.
  - ✓ jQuery pour des échanges Ajax, gestion des formulaires.



# INITIATION À ES6/2015

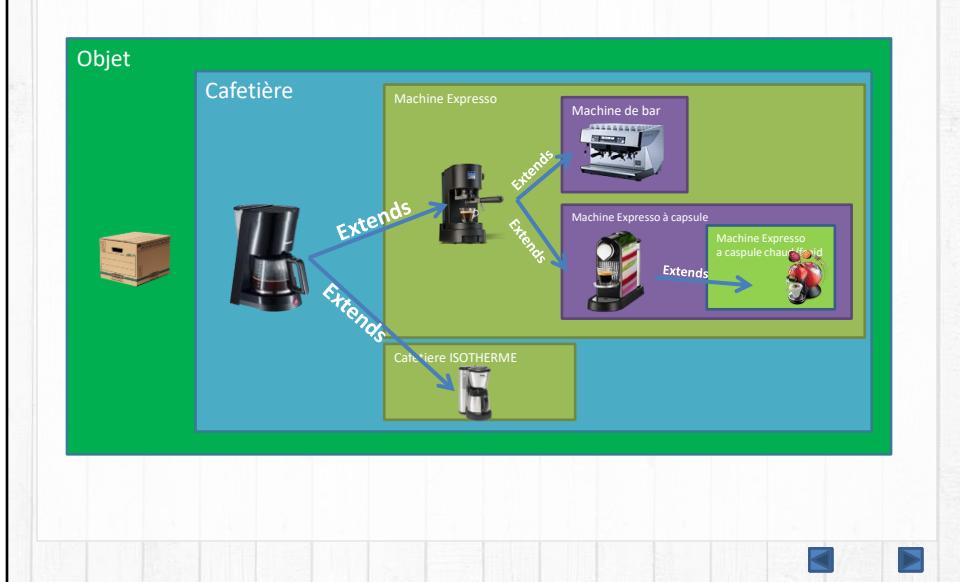


## Sommaire

- ❑ Les nouveautés principales.
- ❑ Les superset JavaScript : TypeScript, Babel, Traceur.
- ❑ ES6, ESNext en production.



## Notion d'objet



## ECMA-262 timeline

- La société Netscape à besoin d'un langage pris en charge dans Netscape Navigator pour rendre plus dynamique ces contenus
  - En 1995 le javascript 1 est née, le nom est choisie pour volontairement créer la confusion avec java
    - Normalisé en juin 1997 sous a norme ECMA-262
  - L'es4 sera abandonné au profit de la branche concurrente es3.1 et donnera la version es5 en décembre 2009
  - L'es6 ou ES2015 subira une nouvelle version tous les ans
    - ES2016, ES2017, ...
- La prise en charge de ces nouvelles version ira moins vite que la fréquence de publication de nouvelle version



## ES6 variables

- 2 nouveaux opérateur d'instanciation de valeurs existent en es6

- **const**

- Les **const** possédé une **référence constante**

- L'objet ciblé par la ce nom de variable ne peut changer
      - Le contenu de l'objet peut être modifié
      - Les fonctions qui sont invariables sont à déclarer comme const

- **let**

- Une variable qui peut être affectée par diffèrent objet ou valeur au cours de sa durée de vie
    - La référence mémoire de ce nom peut être affecté à différentes évaleurs au cours de sa vie
    - Son contenu peut lui aussi varié durant sa durée de vie



- **Getter / setter**

```
animal.js
class Animal {
    set eats(food) {
        this.food = food;
    }
    get dinner() {
        return `eats ${this.food || 'nothing'}`;
    }
}
```

- **Usages :**

```
let animal = new Animal();
animal.food='oak';
console.log(animal.dinner);
```



## Extension avec class

- L'extension de class est simplifié :

```
└─ extends.js
    class Animal {
        constructor(name='Anon'){
            this.name=name;
        }
    }

    class Lion extends Animal{
        constructor(name='Lion'){
            super(name);
        }
    }
```



## ES6 arrow functions

- Il est possible de déclarer une fonction en préservant this sur l'instance dans lequel elle est déclarer
  - On parle d'arrow function

```
const maFunction = (arg) => {...}
```

- Une arrow fonction peut n'avoir qu'une seul instruction qui sera retourne grace à cette syntaxe :

```
const maReturnFunction = (arg) => instruction()
```



## Es6+/ts spread operator

- Il est possible de déverser tout le contenu d'un conteneur dans un autre (obj vers obj / array vers array)

- Déverser une array dans une autre

```
const tab1=[{key1:'value1'},{key1:'value2'}]  
const tab2=[ ...tab1];
```

- Déverser le contenu d'un objet dans un autre objet

```
const obj1={key1:'value1',key2:'value2'};  
const obj2={ ...obj1, key1:'valeur3'};
```

Des  
ES2018/ES  
9

En es5 il était possible de faire la même grâce à Object.assign

Pour le spread des objets si une clef est redefinit post spread operator, la dernière valeur définit sera celle qui sera préservé dans e container final

## Type script

- Du javascript en mieux
  - Un langage écrit par Microsoft
  - Sortie en version 0.8 en Octobre 2012
  - Dernière version 3.8.3 du 28 fevrier 2020
- Proche du **js** et du **c#**, **java**
  - Cocreer par le principal inventeur du c# & concepteur du .NET
  - Interface, module, class, héritage, ...
- Un vrai langage Objets avec typage fort, intégration forte dans l'ide et dans **node.js**
- Un outils de trans-compilation
  - Du **ts** qui vers du **js** grâce à une commande « **tsc** »
- Un outil pour le **ts**
  - Un Linter « **tsLint** » permet le **marquage d'erreurs** et la **standardisation de l'écriture** du code



## Typage

- Typage basiques :

```
let varName: Type;  
let nombre: number;  
let chaine: string;
```

- Le Duck Typing

```
let departement = 56; //attention 03 allier  
let fullName = `Sarah Vigote`;
```

- Rendre une variable facultative

```
let departement?: string; //attention aux accès
```

Les types: doc : <https://www.typescriptlang.org/docs/handbook/basic-types.html>

1. Boolean
2. Number
3. String
4. Array
5. Tuple
6. Enum
7. Any
8. Void
9. Null and Undefined
10. Never
11. Object

## Typage arrays et autre generiques

- Les génériques

- Fonction nécessitant un type de quelle doit manager
- Exemple d'un objet Array qui limite le type de contenu a des string
  - » Les fonctions de l'array vont définir de façon générique que si que des string sont stocker il peut renvoyer que des string
  - » <T> pour dire un type générique pas connus lors de l'écriture d'une fonction
    - et dépendra du type fournit lors de l'usage a chaque appel d'usage

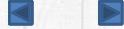
```
let names: Array<string>;
```

- Le typage des *array* possède plusieurs syntaxes:

- Il possède aussi les syntaxes suivante en plus de la syntaxe générique

```
let names string[];
```

```
let names: [string];
```



# Typage

- L'usage de let et const
- Typage basiques de variable:

```
let varName: Type;  
let nombre: number;  
let chaine: string;
```

- Typer pour de fonction

```
function hello(  
    nom: string,  
    age: number = 25,  
    isMale?: boolean  
): string  
{return `Demat ${nom} ${age}`;}
```

La fonction doit renvoyer une string

nom est un paramètre typés obligatoire

age est un paramètre typés obligatoire  
Mais avec une valeur par défaut

isMale est un paramètre facultatif  
sans valeur par défaut

Les types: doc : <https://www.typescriptlang.org/docs/handbook/basic-types.html>

1. Boolean
2. Number
3. String
4. Array
5. Tuple
6. Enum
7. Any
8. Void
9. Null and Undefined
10. Never
11. Object

## esnext

- On parle d'es next toutes les version post es6 (es2016)
- Liste de apports de chaque versions :  
<https://www.javascripttutorial.net/es-next/>
- Chaque nouvelle version propose des légers ajout pour faciliter le développement
- Seul node supporte nativement une grande partie de la spec ESNEXT :  
<https://node.green/>



## Pour assurer la construction

- Webpack

- L'assembleur de module
  - Ex Avec common.js → require("")
  - avec import bar from './bar';
- React/cli nous fourni un environnement
- webpack préétabli



- Babel.js

- Se dit :le « compilateur » de js
  - Il compile le code et peut le rendre dans des
  - Versions ES antérieurs
  - Transpile aussi le ts en js

**BABEL**

Web pack : <https://webpack.js.org/>

Babel : <https://babeljs.io/>

## transpileur

- Pour des raisons de compatibilité, l'es6, esnext, le typescript, et bien d'autres *superset*

- Doivent être transformé en du code compréhensible

- On parle de transpileur

- Compilation et transformé en version différente



## Conclusion

❑ Les nouveautés principales.

- ✓ Les superset JavaScript : TypeScript, Babel, Traceur.

❑ ES6, ESNext en production.



# **PROGRAMMATION ORIENTÉE OBJET**



## Sommaire

- ❑ Rappels sur l'objet. Propriétés.
- ❑ La chaîne de portée. La chaîne de prototype.
- ❑ Méthodes. Héritage. Portée des données privées et publiques. Mapping et sérialisation.
- ❑ Design Pattern Clés en JavaScript.
- ❑ Closure function. Singleton et Modules.



## Public / Privé

- Tous les champs ne sont pas forcément accessible depuis l'extérieur de l'instance de l'objet

### Privé

- Fonctions
  - Digérer
  - Interpréter le signal de l'œil
- Valeurs
  - Taux de sucre dans le sang
  - Image saisie par l'œil
  - Consommation d'oxygène
  - ...

### public

- Fonctions :
  - Manger
  - Regarder
- Valeurs
  - Température \*(read only),
  - Epreinte digitale,
  - Poids,
  - Taille de chaussure \*,...

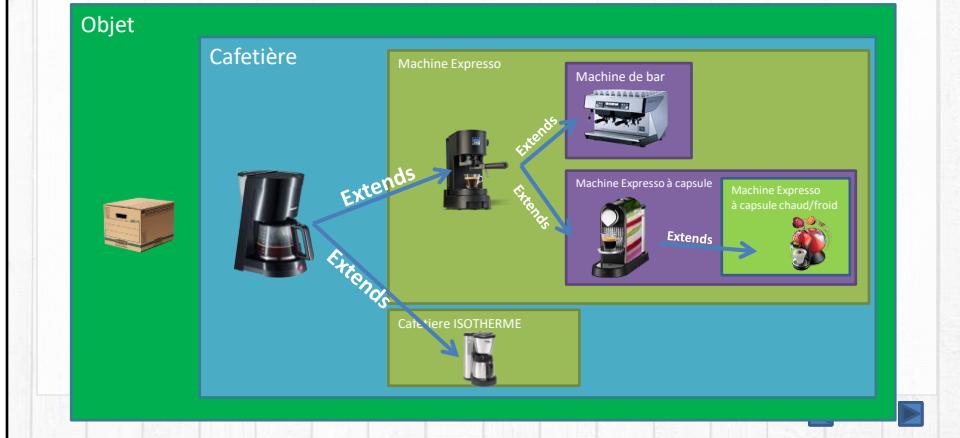


## Notions d'objet

- L'écosystème Objet, un programme peut être conçu comme un écosystème rempli d'entités en interaction (objets), disposant de caractéristiques propres (propriétés) et de moyens d'action (méthodes) et répondant à des stimuli externes (événements).
- On appelle "objet" chaque instance d'ensemble de valeurs regroupées selon une méthode(class/function);
  - Une machine à laver 1500 tour/min d'après le plan de construction du constructeur
- On appelle
  - "propriété" une variable
  - "méthode" une fonction de la classe
    - Le nb de tours/min est une des propriétés. :
    - Programmer le cycle de lavage est une méthode .
- On appelle gestionnaire d'événement une fonction qui répond à une action ou interaction.
  - La machine est finit elle bip pour signaler votre action à faire une fois la machine achevé

## Extension et héritage

- Chaque objet peut en tendre un autre et hérite de son parent
  - class maClass extends autreClass{}



## Objet en JS

- En **ES5** les *class* n'existent pas encore
  - Les objets sont des **fonctions** qui (si instancié avec **new**) persisteront en mémoire pour rester accessible

Exemple.js

```
function MonObjet(){...}  
var monObjet = new MonObjet();  
monObjet.property1=5;
```

- En **ES6** le mot clef **class** existe

Exemple.js

```
class MonObjet{}  
const monObjet = new MonObjet();  
monObjet.property1=23;
```



## Privé / Public

- Définition des protées public privé
  - Es5 :

Protées.js

```
function MonObjet(){
    this.publicValue='une valeur';
    this.publicFunction=function(){...}

    var _privateValue='valeur privée';
    function _privateFunction(){
        _privateValue='Valeur Modifiée';
    }
}
var monObjet=new MonObjet();
monObjet.publicValue='Nouvelle valeur public';
```

- Seul les champs portés sur l'instance **this** sont public est accessible de l'extérieur de l'objet
  - Les champs privés sont disponible que dans l'objet et ses enfants



## prototype

- en js il est possible de surchargé à la description de l'objet des champs public

☰ Prototype.js

```
function MonObjet(){...}
```

```
var monObjet1=new MonObjet();
```

```
MonObjet.prototype.nouvelleFonction=function(){...}
```

```
monObjet1.nouvelleFonction();
```



## Portées des variables ES5

- Portée global
  - Définit une valeur, fonction en dehors de tout context dédié et accessible dans toute l'appli
- Portée classe les enfants accèdent aux valeurs du parent

```
function Parent(){  
    var parentValue='un parent';  
    console.log('--Parent--');  
    function Enfant(){  
        var enfantValue='un enfant';  
        console.log('--Enfant--');  
        console.log('\tparent :', parentValue);  
        function PetitEnfant(){  
            console.log('--Petit enfant--');  
            console.log('\tgrand parent :', parentValue);  
            console.log('\tparent :', enfantValue);  
        }  
        var petitEnfant=new PetitEnfant();  
    }  
    var enfant= new Enfant();  
}  
var parent=new Parent();  
  
--Parent--  
--Enfant--  
parent : un parent  
--Petit enfant--  
grand parent : un parent  
parent : un enfant
```

## Es6 et les modules

- ES6 supporte un nouveau fonctionnement
- Décomposition en modules importable
- `<script src="x.js" type="module">`
- La porté du des expression du module sera limité au module
  - Tout ce qui doit être accessible doit être exporté



## Conclusion

- ✓ Rappels sur l'objet. Propriétés.
- ✓ La chaîne de portée. La chaîne de prototype.
- ✓ Méthodes. Héritage. Portée des données privées et publiques. Mapping et sérialisation.
- ✓ Closure function. Singleton et Modules.



## **EXPRESSIONS RÉGULIÈRES**



## Sommaire

- ❑ Structure et syntaxe d'une expression régulière.
- ❑ Validation asynchrone.



## PCRE

- Perl Compatible Regular Expression
  - Fortement intégrer dans les langages informatiques
  - Communément appelé regex
  - Déclaration en js :
    - Objet → `const r=new RegExp('\\\\w*');`
    - forme littérale : `const r=/\\w*/;`
- Description de chaque caractères d'un chaine
  - Par plages de valeurs → [a-zA-Z] ou [0-9]
  - Par raccourcies de plages définit → \w ou \d
  - Par occurrences de plages → \w\* ou \w+ ou \w{2,}
  - Par groupe conditionnel → (A|B)|\w



## PCRE Syntaxe

- Plages de valeurs

- Délimités la plage pour UN caractère:
  - Début de chaîne & fin de chaînes
    - ^
    - \$
- La plage de valeur & caractère constants:
  - A-Z définit tous les caractères entre A & Z
    - » A, B, C, D, E, F, G, H, ...
  - c-g définit tous les caractères entre c & g
    - » c, d, e, f, g
  - 0-9 définit tous les caractères entre 0 & 9
    - » 0, 1, 2, 3, 4, 5, ..., 9
- Exemple :

[B-Gc-fx]-123-[0-5]\$



## Quantificateurs

- Quantificateurs simples :
  - Optionnel : ?
  - 1 à n : +
  - 0 à n : \*
- Quantificateurs complexes :
  - $x = \text{min occur}$ ,  $y = \text{max occur}$
  - Exact : {x}
  - minimum :{x, }
  - min & max : {x,y}



## Plages prédefinis

- Les plages prédefinis :

Whitespace & non whitespace

\s                  \S

Digit & non digit

\d                  \D

Word & non word

\w                  \W

Newline,        tabulation,    null

\n                  \t                  \0



## ECMA & regex

- Recherche simple
  - Regex.exec
- Recherche dans les groupes
  - Pour chaque groupe identifié
    - Nommer les groupes
      - (?<groupName>\w)
    - Trouver par groupName
      - m.groups.groupName

```
const regex = /(\w[\w\d._-] {1,} \w)@(\w {1,}).(\w {2,7})/;
const str = `desorbaix@free.fr`;
let m;

if (regex.exec(str) !== null) {
  console.log(`\$ {str} format \$ {regex} OK`);
}

if ((m = regex.exec(str)) !== null) {
  m.forEach((match, groupIndex) => {
    console.log(`Found match, group \$ {groupIndex}: \$ {match}`);
  });
}
```

Exemple de récup des groupes nommés :

```
const regex = /(?<user>\w[\w\d._-] {1,} \w)@(?<domain>\w {1,}).(?<firstLevel>\w {2,7})/;
const str = `desorbaix@free.fr`;
let m;
if ((m = regex.exec(str)) !== null) {
  console.log(`user : \$ {m.groups.user}\tdomain : \$ {m.groups.domain}\tfirst level dom : \$ {m.groups.firstLevel}`);
  m.forEach((match, groupIndex) => {
    console.log(`Found match, group \$ {groupIndex}: \$ {match}`);
  });
}
}
```

## Regex flags

Les flags spécifient les conditions de recherche et non les valeurs à chercher

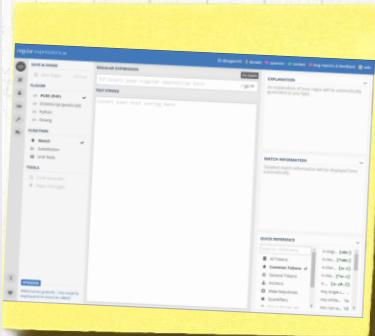
- **Globale : /g**
  - Recherche récursive dans toute la chaîne
- **Multilignes : /m**
  - Recherche dans toutes les lignes
    - Début et fin signifiés par ^ et \$
- **In sensitive : /i**
  - Insensibilité à la casse dans la chaîne
- **Single line : /s**
  - S'arrête à la fin de ligne



## ressources

- **Regex101**
  - Ecriture de PCRE
    - Perl Compatible Regular Expression
    - Quick reference guide
  - Test en ligne de regex
  - Génération de codes :
    - Ecma js
    - C#
    - Go
    - Php
    - ...

<https://regex101.com/>



## Conclusion

- ✓ Structure et syntaxe d'une expression régulière.
- ✓ Validation asynchrone.



## **ENVIRONNEMENT DE DÉBOGAGE**



## Sommaire

- ❑ Chrome Devtools, Firefox Developper, Node-debug
- ❑ Gérer les messages d'erreur client en production.
- ❑ Tests multinavigateurs
- ❑ Stockage des tests.
- ❑ Framework de test (mocha, jasmine, Jest, etc.)
- ❑ Tests de performance et de charge.



## Chrome devtools

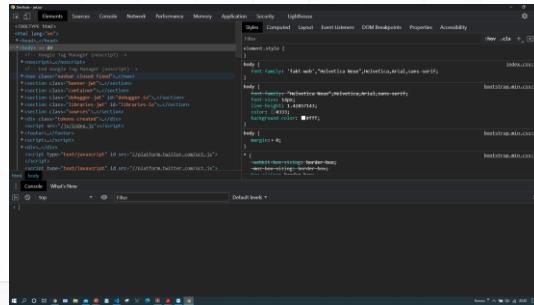
- Chrome propose une série d'outils lié au développement web

Ctrl + Maj + i ou F12

Permet l'ouverture de cette fenêtre

- Propose

- un debugger
- Une inspection
  - Cpu
  - Réseaux
  - Html/css
  - Des storage, ...



## Tester le temps d'exécution

- Console nous fournit 2 fonctions pour mesurer le temps d'exécution
  - console.time('nomDeTache');
  - console.timeEnd('nomDeTache');

testPerf.js

```
console.time('temps fonction');
mafonction(); // ... ect
console.timeEnd('temps fonction');
```



# profiling

- Console permet aussi d'examiner l'usage processeur

- console.profile('nom du profile');
- Console.profileEnd('nom du profile');

```
Profiling.js
function monProcess() {
    console.profile('nom profile');
    //Lot de taches
    console.profileEnd('nom profile')
    ;
}
```

- **3 CRITÈRES MIS EN AVANT :**

- **CPU JAVASCRIPT**

- **CPU RENDU CSS**

- **MÉMOIRE UTILISER POUR LA PRÉSENTATION DES OBJETS**

## Les tests

- Les tests permettent différentes chose :
  - En fonction d'un fichier de définition (cahier des charges ou spec)
    - Vérifier le bon fonctionnement d'un code js
      - » Validité d'une fonction après de changements dans le code
    - Vérifier le bon rendu de l'exécution d'un code
      - » Bon retour de fonctions en fonction de paramètre definis
    - Le vérifier dans différents contextes
      - » Différents environnements d'exécution



## Les moteurs

- Plusieurs moteurs sont disponibles
  - Mocha
    - Se limitant à orchestrer les tests et s'appuyant sur des bibliothèques de tests au choix des développeurs
    - Bonne prise en charge de nodejs
  - Jasmine
    - Est un simulateur de comportement utilisateur dans un navigateur
    - Propose des délais douaniers pour simuler des interactions client/serveur
  - Jest
    - maintenu par facebook,
    - Facile à « décoller »
    - Privilégié pour react, angular, vue, node, ...



## Karma

- Karma
  - Exécute sur différentes plateformes et clients
  - Exécute des tests à distance
  - Prend en charge les tests écrits de la syntaxe de la plupart des frameworks de tests



## Mise en place de mocha

- Installation de mocha
  - npm install --save-dev mocha
  - S'exécute par : ./node\_modules/.bin/mocha
- Nécessite un répertoire **test** à la **racine** du projet
  - Répertoire qui contient la définition ou *la spec*



## Définition d'un test

- Définition d'un cas vérifiable
  - La fonction : `it('should do something',function(){...});`
    - La callback est le contenu du test
  - La vérification de valeur est faite par des assertions
    - Framework d'assertion non fournit de base avec mocha
      - » Utilisons **assert**: `var assert=require('assert');`
      - *Propose :*
        - `assert.fail([message])`
        - `assert.strictEqual(testValue, valeurAttendu[message]);`
        - ...

```
var assert = require('assert');
describe('test de index',function name(params) {
  //definition d'une spec
  it('should correctly multiply', function(){
    //test core js
    var a=2;
    assert.strictEqual(a * 3,7);
  });
});
```

## Tester un module

- Définition de la spec pour un module
  - Imbrication des describe

```
var Mylib = require('../src/Mylib/Mylib.js');

var assert = require('assert');
describe('Mylib', () => {
    //description du module
    describe('#evolution', () => {
        //definition de la section du test pour la fonction evolution
        it('should give an evolution', () => {
            assert.strictEqual(Mylib.evolution(100,200), -100);
        });
        it('should return round value', () => {
            assert.strictEqual(Mylib.evolution(100,30).toString(), '70');
            assert.strictEqual(Mylib.evolution(30,100).toString(), '-233.33');
        })
    });
})
```

```
**src/Mylib/Mylib.js
module.exports = {
    evolution:function(a,b,factor=2){
        return b==0?0:Math.round(Math.pow(10,factor) * 100 * (a -
b) / a) / Math.pow(10,factor);
    }
}

**test/Mylib.spec.js
var Mylib = require('../src/Mylib/Mylib.js');

var assert = require('assert');
describe('Mylib', () => {
    //description du module
    describe('#evolution', () => {
        //definition de la section du test pour la fonction evolution
        it('should give an evolution', () => {
            assert.strictEqual(Mylib.evolution(100,200), -100);
            assert.strictEqual(Mylib.evolution(100,150), -50);
            assert.strictEqual(Mylib.evolution(100,50), 50);
        });
        it('should handle 0 evolution', () => {
            assert.strictEqual(Mylib.evolution(0,100), -Infinity, 'not return -
Infinity')
            assert.strictEqual(Mylib.evolution(100,0), 0, 'not return 0')
        });
        it('should return round value', () => {
            assert.strictEqual(Mylib.evolution(100,30).toString(), '70');
            assert.strictEqual(Mylib.evolution(30,100).toString(), '-233.33');
        })
    });
})
```

## Tests & fonctions

- Définir la présence d'une fonction

```
it('should wait exist', () => {
  assert.notStrictEqual(Mylib.wait, undefined, 'function Mylib wait not accessible');
})
```

- Définir le résultat à l'intérieur d'une callback

```
it('should wait', () => {
  Mylib.wait(1000, function(result){
    assert.strictEqual(result,18);
  })
});
```

```
Js :
module.exports = {
  evolution:function(a,b,factor=2){
    return b==0?Math.round(Math.pow(10,factor) * 100 * (a - b) / a) / Math.pow(10,factor);
  },
  wait:(time,callback)=>{
    setTimeout(()=>{callback(18);},time)
  }
}

Spec :
var Mylib = require('../src/Mylib/Mylib.js');

var assert = require('assert');
describe('Mylib', () => {
  //description du module
  describe('#evolution', () => {
    //definition de la section du test pour la fonction evolution

    //un scenario d'execution
    it('should give an evolution', () => {
      assert.strictEqual(Mylib.evolution(100, 200), -100);
      assert.strictEqual(Mylib.evolution(100, 150), -50);
      assert.strictEqual(Mylib.evolution(100, 50), 50);
    });
    //un autre scenario
    it('should handle 0 evolution', () => {
      assert.strictEqual(Mylib.evolution(0, 100), -Infinity, 'not return -Infinity')
      assert.strictEqual(Mylib.evolution(100, 0), 0, 'not return 0')
    });
    //...
    it('should return round value', () => {
      assert.strictEqual(Mylib.evolution(100, 30).toString(), '70');
      assert.strictEqual(Mylib.evolution(30, 100).toString(), '-233.33');
    });
  });
  describe('#wait', () => {
    it('should exist', () => {
      assert.notStrictEqual(Mylib.wait, undefined, 'function Mylib wait not accessible');
    });
    it('should wait', (done) => {
      Mylib.wait(1000, function(result){
        assert.strictEqual(result,18);
        done();
      });
    });
  });
})
```

## La commande Mocha

- **Mocha**

- -help
    - Aide de la commande

- --grep ou -g contenu de it ou describe
    - Limite l'exécution du test

- » Ex : '#evolution' ou 'Mylib'

- -R reporter Name
    - Permet d'utiliser une méthode d'affichage pour le résultat du test

- » Ex : list, landing, ...

- -b
    - Arrêt à la première erreur



## mocha

- Déclarer des ***it('message')***
  - Pour obtenir des test prévue mais pas encore décris
- Déclarer des ***it.only('message',function)***
  - *Limite l'action de ce test à ce test uniquement*
- Déclarer des ***it.skip('message',function)***
  - *N'exécute pas ce test mais signal qu'il est skip*



## Créer des script de test

- Dans package.json

- Section script

- Créer une commande pour exécuter votre test

```
"script":{ "test" : "./node_modules/.bin/mocha -R list" }
```

- Créer une commande pour exécuter qu'une partie du test

```
"script":{  
  "test": "...",  
  "test-evolution" : "./node_modules/.bin/mocha -R list -g '#evolution'"  
}
```



## Mocha et l'async

- Les before / after

- `before(function)`

- Exécuter avant le démarrage de la batterie de test
    - `beforeEach` demarrer avant chaque sections

- `after(function)`

- *Exécuter après la batterie de test*
    - `afterEach`



## Etendre les assertions

- La librairie chai
  - Des assert plus avancé
  - Installation  
npm i –save-dev chai
  - Usage  
var asset = require('chai').assert
  - Plus de fonctions d'assert
    - IsFunction(Mylib.wait)
    - isTrue(value), ...
- <http://chaijs.com>



## L'approche should expected

- Grace à require('chai').should
  - Tous les objets possèdent la méthode should
    - » Qui lui-même possède be, equal, have ...
- Grace à require('chai').expected
  - La fonction expected( obj ).to
    - » Qui fournit lui aussi un lot de comparatifs
- <https://www.chaijs.com/guide/styles/>



## Conclusion

- ✓ Chrome Devtools, Firefox Developper, Node-debug
- ✓ Gérer les messages d'erreur client en production.

### Tests multinavigateurs

- ✓ Stockage des tests.
- ✓ Framework de test (mocha, jasmine, Jest, etc.)
- ✓ Tests de performance et de charge.



## ECHANGE DE DONNÉES, INTERACTIONS



## Sommaire

- ❑ Solutions de stockage embarqué de données : SQLite, LocalStorage, Cookies.
- ❑ Compenser les latences de communication.
- ❑ JSON Web Token, sécuriser les échanges.
- ❑ XMLHttpRequest, API fetch, consommation de services distants.
- ❑ Serveur JavaScript Node.js.



## L'event online

- Connaitre l'état de connexion au réseau

- Etat de la carte pas état de la connexion

❑ Online.js

```
window.navigator.onLine ou navigator.onLine
```

- Un event permet de gérer le changement d'état

❑ changeState.js

```
window.addEventListener('offline', (e) => { ... })
```

```
window.addEventListener('online', (e) => { ... })
```



## localStorage

- Un stockage persistant dans le navigateur
- Un couple propriété / valeurs
  - setItem

```
localStorage.setItem('monChat', 'Tom');
```
  - getItem

```
var cat = localStorage.getItem('monChat');
```
  - removeItem

```
localStorage.removeItem('monChat');
```
  - Clear

```
localStorage.clear();
```



Documentation : [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Storage\\_API/Using\\_the\\_Web\\_Storage\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API)  
<https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage>

sessionStorage proche de localStorage

- sessionStorage sera détruit à la cloture de la session navigateur
- localStorage n'a pas de date de peremption

Le fonctionnement est similaire

## Cookies

- Propriété de *document*
  - document.cookie
  - Suppr. à la fermeture du navigateur
  - path par def. = page courante
  - une chaine de caractères
- Ecriture

```
document.cookie = "username=John Doe";
```

```
document.cookie = "username=Nikola_TESLA; expires=Thu, 18 Dec  
2050 12:00:00 UTC; path=/";
```

- Lecture

```
var cookie=document.cookie;
```



# cookies

- Lecture direct

```
var cookie = new RegExp('(; )'+cookieName+'=([^\s;]*')');
```

- fonctions de lecture

```
function readCookie(cookieName) {  
    var re = new RegExp('(; )'+cookieName+'=([^\s;]*')');  
    var sMatch = (' '+document.cookie).match(re);  
    if (cookieName && sMatch) return unescape(sMatch[1]);  
    return '';  
}
```

- fonction d'écriture

```
function setCookie(cookieName,cookieValue,nDays) {  
    var today = new Date();  
    var expire = new Date();  
    if (nDays==null || nDays==0) nDays=1;  
    expire.setTime(today.getTime() + 3600000*24*nDays);  
    document.cookie =  
        cookieName+"="+escape(cookieValue)+";expires="+expire.toGMTString();  
}
```

## SQLite

- L'API webSQL propose une base de données embarquée dans le navigateur :
  - Basé sur le standard SQLite
  - Le SQL un langage de requêtage de base de données

Exemple des commandes SQL : <http://www.debray-jerome.fr/articles/L-api-websql-en-html5.html>

Exemple d'ouverture d'un BDD :

```
var db;  
if(window.openDatabase){  
    db = openDatabase('ma_base', '1.0', 'database',  
2000000);  
}
```

## Base SQL

- La base de données est un rassemblement logique et fonctionnel d'un ensemble de tables
- Création d'une base en SQL

```
CREATE DATABASE nomDeLaDB ;
```

- Suppression d'une base de données

```
DROP DATABASE nomDeLaDB;
```

\* Use nomDeLaDB : cette fonctionnalité est effectuer des le connexion en php par le biais d'une fonction PHP

## Tables SQL

- Les **tables** sont des **rassemblements logiques** des données en fonction de ce qu'elle caractérisent et **leurs occurrence** dans les enregistrements
- Les **associations** ont lieu d'exister **que si** les entités qu'elles relient possède **chacun des occurrences >1**  
(user/acheter/produit, mais pas user/titre)
  - Si une **information** a des **besoins de présence récurrent** dans plusieurs enregistrements on sépare en 2 tables on appelle ça les associations hiérarchiques EX : user(Alexandre, DESORBAIX, ...) et titre(Mr, Monsieur, ...)
  - Dans ce cas la table **user comportera une clef étrangère** pointant vers l'**ID du titre correspondant** au user dans l'enregistrement.



Un user est composé d'**un seul et unique** titre donc pas d'association malgré qu'un titre peut **être porté par plusieurs** users

Un user peut acheter **plusieurs** produits et un produit peut être **acheté par plusieurs** users donc **association obligatoire**

## Tables SQL

- Création d'une table :

```
CREATE TABLE nomTable(nomChampID Type PRIMARY KEY,  
                      nomChamp2 TYPE[,...]  
                      [, nomChamp2 REFERENCES table2(idtable2)]);
```

- Champ **ID est obligatoire** et est préfixer de PRIMARY KEY, il est **fortement conseiller** d'utiliser des **entier** ou des **long**
- Les clefs étrangères peuvent être déclarées inline en référençant le champs de la table portant la clef sur la table et le champs quelle pointe.
  - la bonne pratique est de les déclarer a part avec un nom les identifiants pour facilement les désactiver. Le champ Origine et le champ pointé doivent être du même type
    - Apres la création de la table : ALTER TABLE
- Ajouter les autres champs

Les types principaux des champs d'une table :

CHAR (length) → chaîne de caractères à taille fixe (ex. : codePostal  
VARCHAR(5))

VARCHAR(length) → chaîne de caractères à taille variable avec taille max (ex. : prenom VARCHAR(24))

FLOAT → un nombre à virgule

INT → un entier

DATETIME → une date avec l'heure

## Tables Sql

- Ajout d'un enregistrement dans une table \*

```
INSERT TABLE user(id,nom) VALUES (0,'DESORBAIX');
```

- Lecture des enregistrements d'une table\*

```
SELECT (id, nom, pnom [AS prenom]) FROM user, acheter [A] WHERE id= 1;
```

- Suppression d'un enregistrement\*

```
DELETE FROM user WHERE id = 1 ;
```

- Suppression d'un table\*

```
DROP TABLE user;
```

\***INSERT** :Les enregistrements peuvent être fourni par lot ou les VALUES sont déclarer par ensemble les uns a la suite des autres

\***SELECT** : A : il est possible de sélectionner tous les champs d'une table en remplaçant (champ, champ, ...) par SELECT \* FROM...

B : les champs peuvent être renommé avec nomColonne AS nouveauNom (en bleu : pnom AS prenom)

C : les tables peuvent être renommées avec une abréviation arbitraire pour faciliter les corrélations d'id (en vert : acheter A)

\***DELETE** : si le WHERE n'est pas spécifié tous les enregistrements de la table sont supprimés

\***DROP** : Attention aux clef étrangère qui doivent soit être désactivé pour cette opération ou ne pointant par sur les un id de la table

## Les conditions du WHERE

- La section WHERE d'une requête permet de limiter la portée d'une requête ou de lié des champs
- Il est constituer de
  - De filtrage de valeurs  
user.pnom IN ('Alexandre', Renaud')
  - Filtrage par expression  
user.pnom LIKE "%lex%dr%"
  - Comparaisons :  
= != < > <= >= IS NULL IS NOT NULL
  - Et d'assemblage de plusieurs conditions  
AND / OR
- Des parenthèses peuvent être utilisées pour créer des ensembles de conditions a assembler avec d'autres
- Des calculs peuvent être effectuées pour faire les comparaisons et filtres
- Des sous requêtes peuvent être faites entre ( ) pour filtrer des enregistrements en fonction du résultat d'une autre requête

L'opérateurs de filtre :

**A** : Opérateur peut être présent **une seule fois**

**B** : opérateurs peut être pensent **une à plusieurs fois**

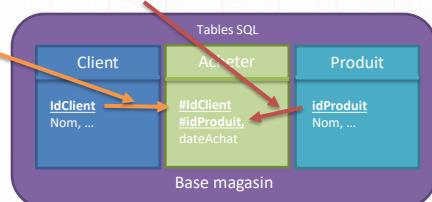
Liste des opérateurs complets : <https://dev.mysql.com/doc/refman/8.0/en/non-typed-operators.html>

## Les conditions du WHERE

- La section WHERE d'une requête permet de limiter la portée d'une requête ou de lier des champs
- Il est constitué de :
  - Jointure de tables :

```
SELECT * FROM client C, acheteur A ,produit P WHERE  
C.idclient = A.idclient AND P.idproduit= A.idproduit
```

- Les jointures peuvent prendre d'autres formes avec **JOIN**, voir la doc



## SQLite

- Ouverture d'une base de données

Websql.js

```
Var db = openDatabase(nom_de_la_bdd(string),
                      version_de_la_base_de_données(string),
                      commentaire_de_la_base_de_données(string),
                      la_taille_estimé(entier), callback(function)
);
```

Exemple des commandes SQL : <http://www.debray-jerome.fr/articles/L-api-websql-en-html5.html>

Exemple d'ouverture d'un BDD :

```
var db;
if(window.openDatabase) {
    db = openDatabase('ma_base', '1.0', 'database',
2000000);
}
```

## webSQL transaction

- Pour effectuer une transaction il existe 2 fonctions

- Pour l'écriture de valeur sans récupération d'info (success est optionnel)

```
db.transaction(function(tx) { ... },  
               function error(e){ ... },  
               function success(e){ ... }  
);
```

- Pour la récupération de valeur depuis la base

```
db.readTransaction(function (tx) { });
```



## webSQL Opération de tables

- Création de table

  └ Createtable.js

```
db.transaction(function(tx) {  
  tx.executeSql(  
    "CREATE TABLE TableTest (id REAL UNIQUE, text TEXT)",  
    [],  
    function success(tx){...} ,  
    function success(tx){...});  
});
```

- Pour les insert, drop, delete, ect le même procédé est utilisé
  - Reste plus qu'à bien construire sa requête SQL

Voici un exemple d'*INSERT* avec des valeurs :

```
db.transaction(function (tx) {  
  tx.executeSql(  
    'INSERT INTO foo (id, text) VALUES (?, ?)',  
    [id, userValue],  
    onSuccess,  
    onError  
  )  
});
```

## webSQL requetage

- Pour le requêtage il faut en plus lire les résultats reçus :

SelectInto.js

```
db.readTransaction(function (t) {
    t.executeSql(
        'SELECT title, author FROM docs WHERE id=?',
        [id],
        function (t, data) {
            for (var i = 0; i < data.rows.length; i++) {
                document.getElementById('log').innerHTML += '<br/>' +
                    data.rows.item(i).author;
            }
        });
});
```



## Visualiser une base WEBSQL

- Dans les outils du devtools chrome / ff
  - Section application
  - Onglet storage / websql

The screenshot shows the Chrome DevTools Application tab interface. On the left, there's a sidebar with sections for Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, ma\_base, TableTest, Cookies), and Cache. The main area displays a table with columns 'rowid', 'id', and 'text'. There are two rows: rowid 1 with id 1 and text 'une donnée', and rowid 2 with id 2 and text 'une autre sdfs donnée'.

	rowid	id	text
	1	1	une donnée
	2	2	une autre sdfs donnée

## JSON webToken

- JWT est un standard (RFC 1519)
  - Il permet l'encodage/décodage d'un flux json
    - Encodé de 256 à 512bits
  - Grace a une clef secrète
  - Un lots d'API pour beaucoup de langages
    - Serveur
    - Appli de bureau
    - Js , ...

<https://jwt.io/>



Rfc : <https://tools.ietf.org/html/rfc7519>

## http requests

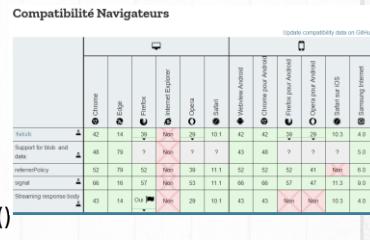
- les appelle http :
  - Opérateur historique
    - XMLHttpRequest
  - API avec promises
    - `window.fetch`

Do I tell him  
fetch() and then()?



## http requests

- Pour faciliter l'accès aux données nous privilégierons le rest
  - Les échanges XML sont plus durs à mettre en œuvre et demande plus de ressources que je le JSON
- En js Historiquement les appelles http(sync et async) étaient fait par xhr (ou XMLHttpRequest)
  - **Xhr**
    - Async grâce à onreadystatechange
    - Prise en charge : fortement compatible
  - **Fetch** Une nouvelle approche est disponible dès es6
    - Approche async promise avec `then()`
    - Prise en charge : moins compatible



## C'est l'histoire d'un mec ...

- Jean-Pierre lorsque il boit son café au bar joue souvent au jeux d'argent.
- Plutôt chanceux, ce jour d'avril il se dit que c'est aujourd'hui
  - Il gratté, il perd ..
  - Il fais un loto
  - Une course de chevaux, allais démarrer
    - Il a donc fais un pari sur un cheval
      - Qui à un look de gagnant (cf photo 1).
    - Il attend devant la course qui débute
    - Son cheval est placé
      - Mais il a coché sans le vouloir le
        - » 8 petit ange des prés (cf photo 2) → dernier
        - Enfin quand il arrivera, si il arrive

photo 1



photo 2



Ici une notion d' asynchronicité aurais pu etre mis en place  
Jp aurais pu attendre boire un petit café tranquillement ou tout simplement revenir demain

Peut etre que tout sa vie reposais sur ce parie, il a donc tout perdu  
Tout le reste de sa vie dépendais de ce paris il a donc attendu car il ne pouvais rien faire d'autre tant qu'il ne savais pas le déroulement de la suite des choses

## C'est l'histoire d'un mec

- Jp sais qu'il a tout joué au tiercé du coup il rentre pu un sous en poche
- A 22h ce jour d'avril, il entend le tirage du loto

— Souvenez vous, jp à joué avant le tierce :

- Il avais fais sa grille

3 – 8 – 21 – 35 – 49

- Enregister son ticket contre un reçu de jeu
  - Il a donc prévu après le tirage de vérifier si il a gagner

» Il à déjà prévu que si il gagnait, il irais encaisser ses gains

- et que c'est un gros lot, il déménagerais à Mulhouse

» Il a aussi prévu que si il gagnait rien, il jetteraient son ticket

- Vue qu'il étais d'humeur gagnante il a jouer et fait d'autre chose
  - C'est la notification de l'appli qui lui à rappeler que le tirage venait d'être fait

- Jp 42 ans en ce jour d'avril avais tous les bon numéro, dans l'ordre du loto.

Chanceux le type



## technique

- Le jeux a gratter n'avais aucune notion d'async
- La course de chevaux na pas été utiliser sous forme d'async
- Notion de promise es6 et **await** function es7/ts
- Le loto quand à lui :
  - Jouer une fonction **async** qui **await** le tirage du soir
  - Une fois que l'action de jouer est fini j' exécute une tache
    - En sachant à l'avance quelle prendrais du temps et que j'aurais bien d'autres taches à effectuer d'ici le résultat de la fonction appeler.
    - Je prévois directement, au moment de l'appel de la fonction, le **retour d'une promise**.
    - Promesse à laquelle je peux définir quoi faire quand l'exécution sera effectuer
      - avec    `.then(retour=>{ ... })`;



## Async / await / Promise

```
• Fichier.js
async function doSomething2() {
  console.log(await doSomethingAsync());
}

function doSomethingAsync() {
  return new Promise((resolve) => {
    setTimeout(() => resolve('I did'), 3000);
  });
}

async function doSomething() {
  console.log(await doSomethingAsync());
  console.log(await doSomething2());
}

console.log('Before');
doSomething();
console.log('After');
```

- Pour être attendu un fonction :
  - Doit soit être déclarer **async**
    - Comporter un(des) **await**
  - Doit retourner une promise
- Une fonction **async** s'utilise soit
  - » Notion d'attente explicite, préfixer par **await**
  - » Pas d'attente explicite, pas de préfixe à l'appel



## Examinons xhr

setRequestHeader permet la définition des header de requête

- 4 étapes

- Nouveau XHR
  - Crédit de l'objet
- Ouverture
  - Définition de:
    - Méthode http
    - Url
- Evènement
  - Définition de l'action pour chaque changement du *readyState*
- Envoie
  - Le corps à envoyé est l'argument d'entrée

- EXEMPLE GET:

```
let xhr=new XMLHttpRequest();
xhr.open('GET', 'http://clanphs.free.fr/');
xhr.onreadystatechange=(evt)=>{
  if(xhr.readyState<4){return;}
  else if(xhr.status!=200){
    console.log(`error ${xhr.status}`);
    return;
  }
  console.log(JSON.parse(xhr.response));
}
xhr.send();
```

Utiliser fetch :

[https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)

## Examinons fetch

- Method GET par defaut
  - 1<sup>er</sup> then : pour convertir un stream de réponse → réponse lisible
    - Différents formats
    - Conversion json ->objet
  - 2eme then : traitement après conversion
  - Un objet de configuration de la demande peut être fournit en 2eme arg de fetch
    - `fetch('https://www.reddit.com/r/javascript.json', { method: 'post', body: JSON.stringify(opts) }).then(...).then(...)`

• EXEMPLE GET:

```
const myImage = document.querySelector('img');
fetch('flowers.jpg')
  .then(function (response) {
    return response.blob();
})
  .then(function (myBlob) {
    const objectURL =
      URL.createObjectURL(myBlob);
    myImage.src = objectURL;
});
```

Utiliser fetch :

[https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)

## Les services Web

- Définition

Un service Web est un programme informatique inter-opérable permettant la communication et l'échange de données sans intervention humaine et en temps réel (Wikipedia). Basé sur des standards et protocoles ouverts (cf. XML)

- Dans la pratique

- Un service Web est une fonctionnalité (ou un ensemble de fonctionnalités)
- Mise à disposition via une interface (ressource) si possible unique
- Un service peut fournir :
  - du contenu (éventuellement dynamique) mis à disposition (ex: liste de produits)
  - une interface d'enregistrement (ajout, suppression, modification dans une base de données)
  - des fonctionnalités métier (ex: calculs de TVA par pays, des taxes d'une fiche de paye...)

- Les formes du service Web

- SOAP (WS-\*), REST (Restful), appel RPC (version plus ancienne)

**Précisions :** Les services web sont le résultat de la mise à disposition d'un service développé par A sur Internet. Il est ainsi possible de les utiliser pour le développement d'un site. Ils permettent par exemple d'inclure un encart avec la météo sur sa page d'accueil sans avoir à le faire soi-même. Il faudra simplement intégrer ce service à sa page. Il est possible d'utiliser les services Web comme des briques de son site.

**REST (*Representational State Transfer*) :** Est une architecture logicielle, utilisée sur le Web pour décharger le serveur qui n'a pas besoin de gérer l'état des transactions (c'est le client qui sauvegarde toutes les variables utiles aux traitements métiers). Dans l'imaginaire des adeptes du Web agile, REST est une alternative à SOAP plus efficace, en fait les deux techniques ne sont pas incompatibles. On peut faire des échanges SOAP selon une architecture REST.

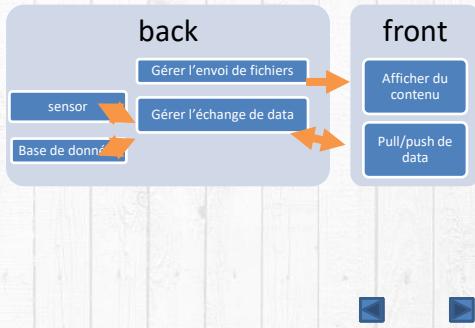
**RPC (*Remote Procedure Call*) :** c'est un appel de procédure distante = qui s'exécute dans un autre espace d'adresse (ordinateur local ou distant), c'est à dire qu'on demande (call) à un ordinateur distant (Remote) de faire une opération (procédure) pour nous et de nous renvoyer éventuellement son résultat (« tout c'est bien passé ». données en retour...).

## Qui fais quoi ?

- Qui à besoin de quoi ?
  - Le client, Un navigateur
    - Charge le site et met en action le JS
    - Il géra grâce au js les interactions IHM
    - Chargerà une version du site html/css/js
      - Et les librairies js associées
  - Il géra les échanges http
    - Accès à des données par Webservice (REST, json)

### – Le serveur

- Il Répondra aux requêtes http pour accéder aux fichiers du site
- Il répondra aux requêtes liées aux données (webservice tel que REST)
- Il gera des échanges internes pour l'accès au sgbdr, aux sensors, ...



## Un web service

- Web Services :

- client,
  - celui qui accède
- fournisseur,
  - celui qui diffuse
- annuaire de services
  - celui qui recense ex UDDI pour les WSDL \*(SOAP)
- intégration d'applications
  - Des API

- Métaphore :

le serveur du restaurant et sa "carte du menu", on se sert pas directement en cuisine



## Un bistro service ?

- Un web service c'est comme  
Un bistrot

- Il est de bon ton d'adopter le protocole adéquat
  - Bonjour
- Un serveur prend ma commande
  - Une bière pression
  - L'hydromel du patron est une bouteille  
Qui ne peut être servis aux clients
- Le serveur me sers ma commande  
si cela est possible.
  - Je consomme ce qui m'a été servi dans
    - le verre adéquat
    - le volume parfait



## Un bistro service?

- Dans certaines enseignes il est possible de savoir ce qui est possible de commander
- Il est même expliquer que
  - Sur présentation d'un coupon Promo une réduction spéciale sera faite



## Un bistrot sur le web?

- Le bar symbolise:
  - Réserve
    - les données, sgbdr, sensor, ...
  - Serveur du bar
    - comprendre une demande
    - Vérifier que le protocole & la demande sont bien formulés
    - Servir, traiter, assembler en fonction de la demande
  - Le client, nous humain,
    - machine consommant ce qui est servie en fonction du contexte
  - Quand à la carte
    - On parle ici de couche de **définition** de ce qui est **mis à disposition** ainsi que le **formalisme** pour le demander



## Protocole http & REST

- Accès CRUD REST

- Create
- Read
- Update
- Delete

- Capacités

- Filtrage
  - champs
  - id
- Positionnement
- Relationnel \*

- Accès à une ressource par HTTP

- Méthodes HTTP
  - POST (create)
  - GET (read)
  - PUT (update)
  - DELETE (delete)

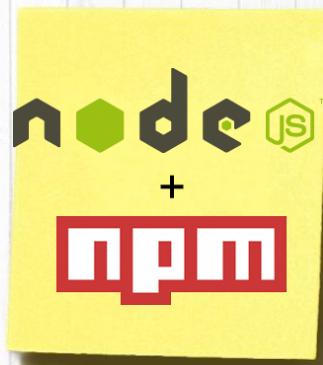
- Erreurs HTTP

- 2xx OK
- 4xx Erreur accès
- 5xx Erreur Serveur

Echanges  
JSON

## Node.js

- Node.js est un moteur JS s'exécutant hors navigateur
  - Il permet d'écrire des apps serveur
  - Il possède toutes les même bases du langage
    - Syntaxe
    - Core functions
  - Il possède un gestionnaire de dépendances
    - NPM



# npm

- Commandes npm

- Initialisation d'un répertoire de projet
  - Créer un fichier minimal package js
    - » **npm init -y**
      - Pré-configure node avec le répertoire...  
(si .git présent)
      - -y : initialisation « quiet »
- Installation des paquets
  - » **npm install -g packetname** packet2@version
    - -g : installation pour l'utilisateur courant,
    - sans -g installation pour le projet, dans le répertoire  
node\_modules
    - --save permet l'ajout du paquet aux dépendances du projet
- Npm start vs npm run monscript
  - Dans package.json, section scripts
  - Script name spécifiques nécessitant pas 'run' ex npm test, npm install, ...



Il existe deux types de dépendances :  
devDependencies  
dependencies



## Server node.js

- Penchons nous sur le module ***express***
  - Permet de fournir un service
    - Ecouteur http
  - Un service serveur
    - Et pouvant proposer des réponses aux requêtes
- Installation :

```
npm install -s express
```



# express

- Usage :

▀ Instantiation du serveur écoutant le port 3000 et répondant à GET /

```
const express = require('express')
const app = express()
app.get('/', function (req, res) {
  res.send('Hello World')
})
app.listen(3000)
```



## Conclusion

- ✓ Solutions de stockage embarqué de données:  
SQLite,  
LocalStorage,  
Cookies.

- ✓ Compenser les latences de communication.

### JSON Web Token, sécuriser les échanges.

- ✓ XMLHttpRequest, API fetch,  
consommation de services distants.
- ✓ Serveur JavaScript Node.js.



# **PROGRAMMATION JAVASCRIPT SOUS HTML5**



## Sommaire

- Nouveaux événements.
- Gestion des API multimédia.
- Modes de communication client/serveur.
- Utilisation des Web Worker.
- Le développement mobile "offline first".
- Optimisation des communications client/serveur.
- Bibliothèques graphiques (D3js, ThreeJS)



## HTML5

- La spécification de html5 renvoie beaucoup à javascript
  - La volonté est d'amener js à un niveau équivalent de performance et de technicité que les langages dédiés aux applications lourdes
    - Apparition de nouvelles API html 5 / js standardisées pour tous les navigateurs
    - Nouvelles problématiques liées à l'évolution du mode de consommation du web
      - Page mobile, gestion du cache de la donnée, video, ...



## Sélection

- La sélection est rendue plus aisée avec

`document.querySelector('#css')` → sélecteur de balise unique

`document.querySelectorAll('.css')` → Sélecteur multi-noeuds

- Une sélection plus rapide que beaucoup de frameworks



## Timer & intervalles

- Le timer permet la programmation de l'exécution d'une callback

- Exécuter q'une seul fois

```
const timer=setTimeout(()=>{},1000)
```

- Annuler : clearTimeout(timer);

- Les intervalles

- Exécuter toutes les N millis

```
const timer = setInterval(()=>{...},1000)
```

- Arrêter : clearInterval(timer);



## Worker

- Les worker est :

- Une tâche de fond
  - Des interactions serveur, des calculs lourds, ...
- Une tâche exécuter en parallèle du programme js
  - Sur un autre thread parallèle

Worker.js

```
var myWorker = new Worker("worker.js");
first.onchange = function() {
    myWorker.postMessage([first.value,second.value]);
    console.log('Message envoyé au worker');
}
```

- il peuvent gérer d'autres worker et include du code grâce à **importScript(url)**



## Worker messaging

- Pour communiquer les worker ont besoin :
  - Il **post** grâce à **postMessage** , et **intercept** grâce à **onMessage**

Worker.js

```
var myWorker = new Worker("worker.js");
first.onchange = function() {
    myWorker.postMessage([first.value,second.value]);
    console.log('Message envoyé au worker');
}
myWorker.onmessage = function(e) {
    result.textContent = e.data;
    console.log('Message reçu du worker');
}
```

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

## d3.js

- Data driven document
  - Document driver par des données
  - Manipulation de DOM en fonction de données
- Une library permettant
  - Une sélection et manipulation plus simple du DOM

```
d3.select("body").style("background-color", "black");
d3.selectAll('p').style("color", "blue");
```

```
var svg = d3.select("#svg1").append("svg");
svg.append("rect").attr("x", 20).attr("y", 20)
```



## D3 Transitions

- Il propose :

- Des transitions :

```
d3.select("body").transition().style("color", "black");
```

- De la surcharge de datas pour dynamiser la construction

```
svg.selectAll(".node")
  .data(dataToDrive) // 2
  .enter() // 3
  .append("rect") // 4

  .attr("x", dataDriven => dataDriven.position * 30) // 5
```

<https://www.datavis.fr/index.php?page=firststep>

## Three.js

- Remplaçant JS de papervision3D(flash)
- Supporte le canvas ou le webgl
  - Canvas → sans calcul & accélération du GPU
  - WebGL → avec calcul de rendu GPU



## WebGL

- L'OpenGL est un standard de rendu 3D
  - Profite de l'accélération matériel des GPU
- WebGL est un standard web pour faire de la 3D basé sur OpenGL
  - Fournit par HTML5
  - Faire rendre le dessin par OpenGL
  - Rempli un espace de dessin avec le rendu
  - Supporté dès :
    - FF4, Chrome 9, safari\*, IE11



White/black list GPU : <https://www.khronos.org/webgl/wiki/BlacklistsAndWhitelists>

## Les nouveaux events html5

- Liste des nouveaux events :

<http://41mag.fr/liste-des-balises-html5/liste-des-evenements-html5>

- onerror
- ondrag , ...
- onplay, onpause
- ononline, onoffline
- ...



## Three.JS

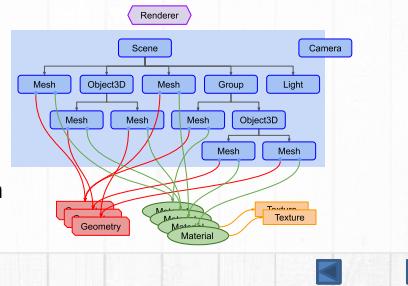
- Three.js est une librairie d'intégration de contenu 3D webGL dans une page web
- THREE propose un renderer
  - Elle dispose de 2 éléments principaux

- Scène

- Constitué d'objets 3D

- Camera

- Pour cibler la visualisation



## Three.js scène & camera

- Initialiser une scène et la camera :

```
Scene.js
// Create an empty scene
var scene = new THREE.Scene();

// Create a basic perspective camera
var camera = new THREE.PerspectiveCamera(75,
    window.innerWidth / window.innerHeight,
    0.1, 1000);
camera.position.z = 4;

// Create a renderer with Antialiasing
var renderer = new THREE.WebGLRenderer({ antialias: true });

// Configure renderer clear color
renderer.setClearColor("#000000");

// Configure renderer size
renderer.setSize(window.innerWidth, window.innerHeight);

// Append Renderer to DOM
document.body.appendChild(renderer.domElement);
```



## Three.js

- Les Mesh
  - Sont composés de géométrie lié à un matériaux
- Définir une forme JS simple :

Scene.js

```
// Create a Cube Mesh with basic material
var geometry = new THREE.BoxGeometry(1, 1, 1);
var material = new THREE.MeshBasicMaterial({ color: "#433F81" });
var cube = new THREE.Mesh(geometry, material);

// Add cube to Scene
scene.add(cube);
```



## Intégration d'un .OBJ

- Vous pouvez intégrer des objets dont les géométries sont déjà définies :



## Conclusion

- ✓ Nouveaux événements.
- ✓ Gestion des API multimédia.
- ✓ Modes de communication client/serveur.
- ✓ Utilisation des Web Worker.
- ✓ Le développement mobile "offline first".
- ✓ Optimisation des communications client/serveur.
- ✓ Bibliothèques graphiques (D3js, ThreeJS)



## Bibliographie

- Livres de référence
  - « XSLT and XPath 2.0 » de Michael Kay chez Wrox
  - « Xml cours et exercices » de Alexandre Brillant
- Sites de référence
  - [http:// www.w3.org](http://www.w3.org)  
(site du W3C - standards)
  - [http:// www.w3schools.com](http://www.w3schools.com)  
(Web programming)
  - <http://www.oreillynet.com/>  
(forum de développeurs)
  - <http://xmlfr.org>  
(blog XML)
  - <http://www.zvon.org>  
(guide de la galaxie XML)



## Conclusion finale

- Remerciements et encouragements

Pendant cette formation, nous espérons :

- que vous avez appris de nombreuses choses théoriques et pratiques
- que vous y avez trouvé plus que ce que vous étiez venu chercher
- et que vous avez passé un bon moment avec votre professeur

Merci d'avoir fait confiance à ORSYS, à bientôt pour d'autres formations !

A BIENTÔT



**ORSYS**  
Informatique et Management

FIN

## **ANNEXES**



*STRUCTURER + AFFICHER DU CONTENU*

**HTML**



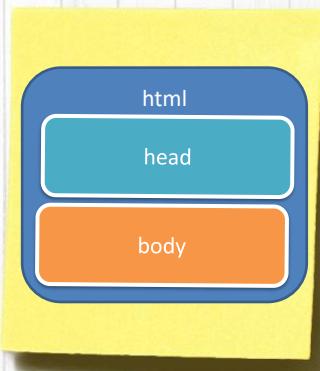
**HTML**



## LE LANGAGE HTML



- Balise principale
  - html
- Balises enfants
  - head
    - Cerveau de la page, non affichable
  - body
    - corps affichable de la page



## head

- <head> → section logique de la page
- Peux contenir
  - <title/>
    - titre de l'onglet
  - <link rel="stylesheet" type="text/css" href="main.css"/>
    - Feuille de style
  - <script src="main.js"></script>
    - Section de script



## body

- <body> → Corps de la page
- contient toutes les balises représentant du contenu affichable



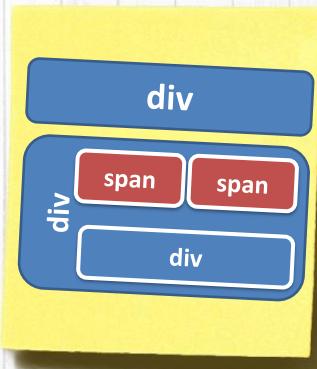
## identifier les balises

- id="IdentifiantUnique"
  - Identifie coté Navigateur css/js (client)
  - Règle **d'unicité globale** à la page
  - Sur toutes les balises a mettre en forme css
- name="nomUnique"
  - Identifie les inputs coté client serveur php/asp
  - présent sur les balises
    - » form
    - » input, textarea, champs de formulaire ...



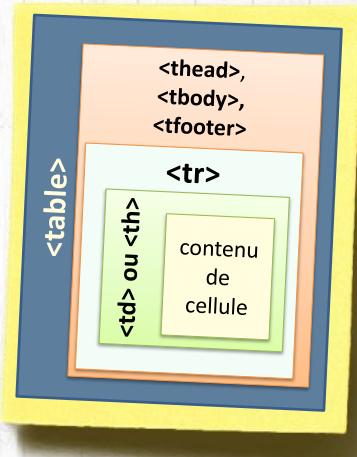
## balises de structures

- <div>
  - un bloc prenant tout l'espace en largeur
  - affecte le flux d'affichage
    - précédent = au dessus, suivant = au dessous
- <span>
  - une zone en ligne pour mettre en forme sans prendre d'espace supérieur au contenu
  - n'affecte pas le flux d'affichage



# Tableaux

- Les tableaux : `<table>`
  - structure
    - `<thead> & <tfooter>` sert à définir des sections de balises explicitement pour l'en tête et le pied de page
    - `<tbody>` sert à définir le corps du tableaux si *tbody et/ou tfooter est définit*
  - Création de lignes
    - » `<tr>` (*Table-row*) : définit une ligne
      - ↳ On y insère un ou des cellules (`td` ou `th`)
        - » `<td>` (*Table-data-cell*) : définit une cellule du données
          - ↳ On y insère un **du contenu html**
        - » `<th>` (*Table-header-cell*) : définit une cellule du typée en tête
          - ↳ On y insère un **du contenu html**
          - ↳ **cellule déjà gras et centrée**



206

Exemple de tableau en html:

```
<table>
  <thead>
    <tr>
      </th>header1</th>
      </th>header2</th>
    </tr>
  <thead>
  <tbody>
    <tr>
      </td>cell1</t2>
      </th>cell2</th>
    </tr>
  <tbody>
  <tfooter>
    <tr>
      </th>footer1</th>
      </td>footer2</td>
    </tr>
  <tfooter>
</table>
```

ou

```
<table>
  <tr>
    </td>cell1</t2>
    </th>cell2</th>
  </tr>
</table>
```

# Tableaux

- Les tableaux :
- Le comportement des cellules de tableaux est configurable pour les fusion de cellules en ligne ou en colonnes avec les attributs suivants sur les <td> & <th>
  - **colspan="nbCol"** sert à définir le nombre de colonnes pour étendre de la cellule horizontalement
    - ↳ On y insère un chiffre
  - **colspan="nbCol"** sert à définir le nombre de colonnes pour étendre de la cellule verticalement
    - ↳ On y insère un chiffre



207

Exemple de tableau en XSL-FO:

```
<fo:table-and-caption>
<fo:table>
  <fo:table-column column-width="25mm"/>
  <fo:table-column column-width="25mm"/>
  <fo:table-header>
    <fo:table-row>
      <fo:table-cell>
        <fo:block font-weight="bold">Car</fo:block>
      </fo:table-cell>
      <fo:table-cell>
        <fo:block font-weight="bold">Price</fo:block>
      </fo:table-cell>
    </fo:table-row>
  </fo:table-header>
  <fo:table-body>
    <fo:table-row>
      <fo:table-cell>
        <fo:block>Volvo</fo:block>
      </fo:table-cell>
      <fo:table-cell>
        <fo:block>$50000</fo:block>
      </fo:table-cell>
    </fo:table-row>
    <fo:table-row>
      <fo:table-cell>
        <fo:block>SAAB</fo:block>
      </fo:table-cell>
      <fo:table-cell>
        <fo:block>$48000</fo:block>
      </fo:table-cell>
    </fo:table-row>
  </fo:table-body>
</fo:table>
</fo:table-and-caption>
```

FOP gère mal le table-and-caption et seul l'ensemble contenu du nœud table est nécessaire  
Nous n'utiliserons donc que table

# Les formulaires

## Structure des formulaires

- La balise `<form>` englobe tous les champs du formulaire
- L'attribut `action`
  - url de la page serveur qui réceptionne le formulaire
- L'attribut `method`
  - GET : envoie le contenu du formulaire condensé dans l'adresse url précédé par ? et séparés par &
    - a.php?champ1=value&champ2=val
  - POST : envoie dans le contenu du paquet http

```
<form action="a.php"
method="GET|POST">
<input name="" type="">
<textarea name="">
</textarea>
<input type='reset' />
<input type='submit' />
</form>
```

détail : [https://www.w3schools.com/tags/tag\\_form.asp](https://www.w3schools.com/tags/tag_form.asp)  
attributs

+target="\_self | \_blank | \_top | \_parent | iframeName "  
    \_self : dans la même page  
    \_blank : nouvel onglet  
+accept-charset=""  
+autocomplete=" on | off "  
+enctype=""  
    application/x-www-form-urlencoded  
    multipart/form-data  
    text/plain

# Les formulaires

## Structure des formulaires

- Différents types d'input
  - **html 4.01** : text, checkbox, file, radio, hidden, password, textarea\*
  - **html 5** : number, range, search, tel, mail, url, week, color, month, image, date, datetime
- 2 input spéciaux :
  - reset → remet tous les champs à la valeur présent dans value=""
  - submit

```
<form action="" method="">  
  
<input name="" type="/">  
  
<textarea name="">  
  </textarea>  
  
<input type='reset' />  
  
<input type='submit' />  
  
</form>
```

détail : [https://www.w3schools.com/tags/tag\\_input.asp](https://www.w3schools.com/tags/tag_input.asp)

## Les listes

- Les listes sont imbriquables
- Liste simple sans puce
  - <**dl**> → définition de liste
  - <**li**> → un item de liste
- Listes de puces simples
  - <**ul**> → Définition de liste à puces
  - <**li**> → Un item de liste
- Liste de puces numérotées
  - <**ol**> → définition de liste numérotées
  - <**li**> → Un item de liste



## balises courantes

- images
  - <**img** **src**="img.jpg" **alt**="texte alternatif"/>
- liens
  - <**a** **href**="http://lien.fr/">zone cliquable</a>
- Saut de ligne
  - <**br**/>
- Gras, souligné, grand, petit
  - <**uu**>
  - <**b**>bold</**b**>
  - <**big**>grand</**big**>
  - <**small**>petit</**small**>



## Conclusion

- Dans cette partie vous avez découvert :

- La structure d'un fichier HTML
- Différentes balises de structures du HTML



- Définition
- Structure
- Formes basiques
- Formes complexes
- Groupes
- Texte, liens, symboles, images
- Filtres
- Textures
- Animations
- Multi formats

## SVG

## Définition

- Scalable Vector Graphics (SVG)
  - Le SVG, langage de graphiques vectoriels, sert à générer des graphiques dynamiquement, à les animer et à les rendre dynamiques.
  - C'est un standard du W3C basé sur XML
  - Utilise les styles CSS pour la mise en forme
  - Contrairement à JPEG ou PNG, il n'est pas trame
  - Son principal concurrent est aujourd'hui Flash
  - Il fonctionne correctement sur les navigateurs récents (plugin MSIE natif MFF)
  - Il existe plusieurs normes SVG, la 1.0 et 1.1 sont les plus utilisées avec une préférence pour cette dernière

## SVG Définition

- Avantages de SVG

- Les graphiques peuvent être liés à d'autres ressources via xlink
- Les SVG sont animés (avec SMIL)
- Et interactifs car scriptable (on peut faire de véritables interfaces)
- Les transformations géométriques sont simples (vectoriel) et s'effectuent sans perte : déplacements, agrandissements, fondus, rotations...
- A un document SVG on peut appliquer plusieurs feuilles de style pour un rendu totalement différent (routes, courbes de niveau, stations, lacs...)
- SVG intègre le DOM (attention pas le DOMHtml !)

## SVG et animations SMIL

- SMIL 2.0 pour les animations
  - SVG intègre la syntaxe SMIL 2.0 pour la gestion des animations

## SVG Définition

- Inconvénients de SVG

- Les données XML sont verbeuses, les fichiers SVG sont de grande taille mais avec une compression Zlib,
  - » le format est plus léger que ses concurrents
  - » lisible nativement compressé



<https://zlib.net/>

217

HOW TO : [https://zlib.net/zlib\\_how.html](https://zlib.net/zlib_how.html)

ZLIB : <https://zlib.net/>

## Structure SVG

- Structure SVG

- Le fichier porte l'extension ".svg" et sa DTD est :

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

- La racine des documents SVG est **<svg>**

- Le xmlns est obligatoire: <http://www.w3.org/2000/svg>

- **viewBox** définit la fenêtre de contenu

- **width** et **height** définissent la taille du SVG dans la page

```
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
viewBox="0 0 500 500">
```

- Certains navigateurs nécessitent de préciser la version="1.1"

- La section <defs> sert aux définitions (variables, effets, symboles...)

### Exemple simple de code SVG :

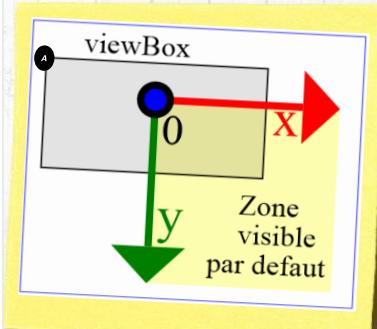
```
<svg width="200" height="200" viewBox="0 0 500 500" version="1.1">
<title> exemple SVG </title>
<desc> Ceci est un exemple de SVG de base </desc>
<defs></defs>
</svg>
```

## svg & css

- La balise `<style type="text/css">`
- capacité `@media`
- sélection CSS classique  $\frac{1}{2}$  &  $\frac{3}{3}$ \* si navigateur compatible
  - » `#premierGroup rect{ ... }`
- propriétés css pour déclarer un attribut svg
  - avec le même nom que l'attribut svg
    - exemple :
      - » en css : `fill : red;`
      - » en attribut svg : `fill="red"`

## SVG & Système de coordonnées

- Point de référence et viewBox
- '0' références est le coins supérieur gauche
- Pas de d'unité = ordre de grandeur
- viewBox permet d'afficher sur des positions négatives
- **viewBox="Ax Ay width height"**



## Formes basiques SVG

- Le dessin svg est assimilable à un jeu de gommettes avec des formes et des tailles variées permettant une fois assemblées de créer des formes complexe
- Les formes de base sont les suivantes :

```
<circle r="100"  
      fill="url(#MyGradient2)"  
      stroke="black"  
      stroke-width="5"  
      cx="150" cy="150"/>
```



```
<ellipse rx="100" ry="150"  
        fill="url(#MyGradient2)"  
        stroke="black"  
        cx="150" cy="200"/>
```



```
<rect  
    width="150" height="200"  
    fill="url(#MyGradient)"  
    stroke="BLACK"  
    x="100" y="100"/>
```



```
<line x1="150" y1="50"  
      y2="200" x2="50"  
      stroke="url(#MyGradient)"  
      stroke-width="10" />
```



### Formes SVG de base :

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"  
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">  
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"  
xmlns="http://www.w3.org/2000/svg">  
  <title> exemple SVG </title>  
  <desc> Ceci est un exemple de SVG de base </desc>  
  <circle r="100px" cx="350px" cy="350px" />  
  <rect x="50" y="50" width="200" height="100" style="fill:red; stroke:blue; stroke-width:10" />  
  <ellipse cx="100" cy="300" rx="50" ry="100" style="fill:#00ff00; stroke:black; stroke-width:1" />  
  <line x1="400" y1="50" x2="350" y2="300" style="stroke:red; stroke-width:3" />  
  
</svg>
```

## Formes complexes SVG

- Il existe aussi des balises pour des formes non définies ou pour des besoins particuliers:

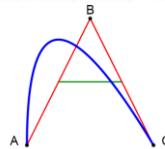
- Le polygone / polylines :** forme pleine coordonnées infini

```
<polygon points="170,110  
140,120 240,230 190,230"  
fill="url(#MyGradient)"  
stroke="black"  
stroke-width="1"/>
```



```
<polyline points="170,110  
140,120 240,230 190,230"  
fill="url(#MyGradient)"  
stroke="black"  
stroke-width="1"/>
```

- Le path est un chemin vectoriel aussi appelé trajet ou simplement vecteur. Il permet d'indiquer des mouvements ou des courbures de texte par exemple.



```
<path  
d="M 100 350 q 0 -500 300 0"  
stroke="blue" stroke-width="5"  
fill="none" />
```

222

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"  
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">  
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"  
xmlns="http://www.w3.org/2000/svg">  
  <title> exemple SVG </title>  
  <desc> Ceci est un exemple de SVG de base </desc>  
  <polygon points="170,170 220,170 220,220 170,220" style="fill:blue; stroke:black; stroke-width:1" />  
  <polyline points="225,225 275,225 275,275 225,275" style="fill:none; stroke:black; stroke-width:1" />  
  </svg>  
  <path style="fill:red; opacity:0.5; stroke:black; stroke-width:1;" d="M100,100 H200  
    C153 334 151 334 151 334  
    C151 339 153 344 156 344  
    C164 344 171 339 171 334  
    C171 322 164 314 156 314  
    C142 314 131 322 131 334  
    C131 350 142 364 156 364  
    C175 364 191 350 191 334  
    C191 311 175 294 156 294  
    C131 294 111 311 111 334  
    C111 361 131 384 156 384  
    C186 384 211 361 211 334  
    C211 300 186 274 156 274" />
```

### Remarques :

Notez le code de l'expression d (data) de <path>, il s'agit de coordonnées de courbes de Bézier (définition de points et tangentes) !

## Remplissage et traits

- Remplissage *fill*

- Couleur css

`fill="LIGHTBLUE"`



- Gradient svg

`fill="url(#gradientId)"`



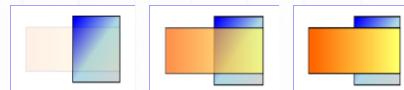
- Opacité

`opacity="value"`

0.1

0.7

1



- Bordures *stroke*

- Couleur

`stroke="BLACK"`



`stroke="#2E8B57"`



- Forme

`stroke-dasharray="value"`

"40,10"



"30,5,10,20"



- Opacité

`stroke-opacity="value"`



- Epaisseur

`stroke-width="value"`

223

## Les Groupes SVG

- Le groupe **<g>**

- On peut regrouper certains éléments à l'aide de la balise **<g>** (group) pour des raisons de facilité

- de déplacement
  - d'application d'effets
  - de sélection de groupe

👉 Mettre un **id** et/ou une **class** pour l'identifier au css

## Texte, liens, symboles, images

- Textes
  - On peut utiliser la balise **<text>** pour insérer du texte
- Liens
  - On peut créer des liens à l'aide de xlink (pas seulement des liens )
    - **<a xlink:href="..."** avec **<svg xmlns:xlink="http://www.w3.org/1999/xlink" ...**
- Symboles
  - Pour gagner en temps (load) et poids des fichiers on peut utiliser des symboles
  - On définit un symbole dans **<defs>** avec **<symbol id="mysymbol">** et **<g>**
  - On l'utilise avec **<use xlink:href="mysymbol" x="..." y="..." />**
- Images
  - On peut insérer des images avec **<image xlink:href="myimage.jpg" ... />**

### Exemple de création et d'utilisation de symbole :

```
<defs>
  <symbol id="croix" height="25" width="25">
    <g fill="red" stroke="none">
      <rect x="10" width="10" height="25"/>
      <rect y="10" width="25" height="10"/>
    </g>
  </symbol>
</defs>

<use xlink:href="# croix " x="150" y="100"/>
<use xlink:href="# croix " x="250" y="0"/>
<use xlink:href="# croix " x="300" y="300"/>
```

# Textures

- Textures
  - Pour remplir les formes avec des motifs évolués, SVG propose les gradients
    - **linearGradient** dégradé linéaire
    - **radialGradient** dégradé radial
  - On les définit dans la partie <defs>
  - On l'applique sur les objets à l'aide de l'attribut **style="fill:url(#mygradient);"**
- Rappel RVB (RGB)
  - RVB = Rouge Vert Bleu (RGB en anglais)
  - Ce sont des spectres d'additions (rouge+vert=jaune et pas vert+jaune=bleu), qui combinées, permettent d'obtenir les couleurs de bases
  - Chaque intensité de couleur va de 0 à 255 (0 = éteint et 255 = allumé au max)

226

## Exemples de gradient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<title> exemple SVG </title>
<desc> Ceci est un exemple de SVG de base </desc>

<defs>
<linearGradient id="lingrad" x1="30%" y1="0%" x2="90%" y2="0%">
<stop offset="0%" style="stop-color:rgb(255,255,0); stop-opacity:1"/>
<stop offset="100%" style="stop-color:#FF0000; stop-opacity:1"/>
</linearGradient>
</defs>

<circle r="100px" cx="350px" cy="350px" style="fill:url(#lingrad); />
<rect x="50" y="50" width="200" height="100" style="fill:url(#lingrad); stroke:blue; stroke-width:10;" />
</svg>
```

## Exemples de codes combinés :

[http://www.w3schools.com/svg/svg\\_examples.asp](http://www.w3schools.com/svg/svg_examples.asp)

## Pour connaître presque tous les éléments SVG :

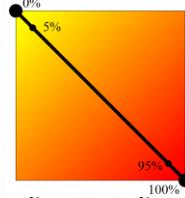
[http://www.w3schools.com/svg/svg\\_reference.asp](http://www.w3schools.com/svg/svg_reference.asp)

# Gradient SVG (dégradés)

- Dégradés

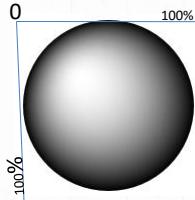
- Transition de couleur et/ou d'opacité est possible grâce aux gradient soit linéaire, soit circulaire (dit radiaux)

```
<defs>
  <linearGradient x1="0" x2="100%"
    y1="0" y2="100%" id="MyGradient">
    <stop offset="5%" stop-color="yellow"/>
    <stop offset="95%" stop-color="red"/>
  </linearGradient>
</defs>
```



- **linearGradient**

- **radialgradient**



```
<defs>
  <radialGradient id="grad1" cx="50%" cy="50%" r="50%" fx="33%" fy="33%">
    <stop offset="0%" stop-color="white" stop-opacity="0"/>
    <stop offset="100%" stop-color="darksilver" stop-opacity="1"/>
  </radialGradient>
</defs>
```

## Filtres SVG

- Filtres

- Les filtres sont des effets appliqués à une forme, par exemple un flou, une déviation, une ombre portée... On note, par exemple, les filtres suivants :

• <b>feBlend</b>	→ mode d'intersection (cf. logiciels de graphisme)
• <b>feDiffuseLightening</b>	→ lumière diffuse
• <b>feFlood</b>	→ remplissage
• <b>feGaussianBlur</b>	→ flou circulaire
• <b>feMerge</b>	→ fusion
• <b>feOffset</b>	→ décalage de coordonnées dx, dy de la forme de base

- On les définit dans la partie **<defs>** avec **<filter id="myfilter"> <feABC />**
- On l'applique sur les objets à l'aide de l'attribut  
**style="filter:url(#myfilter);"**

228

### Exemples de filtres :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg width="500" height="500" viewBox="0 0 500 500" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<title> Filtres SVG </title>
<desc> Ceci est un exemple de filtres SVG ! </desc>
<defs>
<filter id="blured">
  <feGaussianBlur in="SourceGraphic" stdDeviation="5" />
  <feOffset dx="5" dy="5"/>
</filter>
</defs>
<ellipse cx="100" cy="300" rx="50" ry="100" style="opacity:50%;fill:black; stroke:black; stroke-width:1;filter:url(#blured);"/>
<polyline points="225,225 275,225 275,275 225,275" style="fill:none; stroke:black; stroke-width:1;filter:url(#blured);"/>
</svg>
```

# Animations

- Animations

- Pour déclencher des effets visuels, on utilise **<animate>** avec les attributs :
  - "attributeType" permet de préciser sur quel langage on va jouer (ex: CSS | XML)
  - "attributeName" indique quel paramètre on va modifier (ex: **opacity**)
  - "from" et "to" indiquent les valeurs de départ et d'arrivée (ex: 1 à 0)
  - "dur" indique la durée de l'effet (ex: 3s)
  - "begin" pour retarder / synchroniser les animations (ex: 2s)
  - "repeatCount" permet de répéter l'animation un nombre de fois précis (ex: 3) avec les paramètres de l'exemple on aurait un effet de fondu qui se lancerait 3x après 2s
- Pour animer une forme on utilise **<animateMotion>** avec les attributs :
  - "path" permet de préciser le chemin (vecteur, trajet) à suivre en un temps donné par "dur"
- Avec **<animateColor>** on peut aussi passer d'une couleur (from) à une autre (to)
- Et on fait des transformations à l'aide de l'attribut **transform="translate(x,y)"** ou de **<animateTransform>** (ex: **type="rotate | scale | translate | skewX/Y..."**)

229

## Exemple d'effet visuel de fondu :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
  <rect x="20" y="20" width="250" height="250" style="fill:blue">
    <animate attributeType="CSS" attributeName="opacity" from="1" to="0" dur="3s"
repeatCount="3" />
  </rect>
</svg>
```

## Exemple de code d'animation :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
  <g transform="translate(80,150)">
    <text id="TextElement" x="0" y="0" style="font-family:Verdana;font-size:18"> C'est du SVG !
  <animateMotion path="M 0 0 L 80 150" dur="5s" fill="freeze"/>
    </text>
  </g>
</svg>
```

## Multiformats

- Génération multi format : **XHTML/Fo + SVG en XSLT**
  - On remarque qu'avec les possibilités d'**XSLT** et de **SVG** on peut extraire des données XML et créer **dynamiquement** des **graphiques** **SVG**
  - Donc **générer des images** à partir **de données**
  - C'est très utile en particulier pour les **outils statistiques** tels que les outils d'aide à la décision, les agrégations peuvent être représentées graphiquement à la volée
  - De plus, le tout peut être **embarqué** dans une page **xHTML** générée ou **XSL-Fo**, elle aussi **dynamiquement** en **XSL-T**, ceci nécessite quelques précautions, dont :
    - l'utilisation de namespaces pour éviter la confusion des langages utilisés
    - sous MFF, l'usage de certaines extensions ("xhtml" au lieu de ".html" ou ".htm")
    - sous MSIE, le recours à une balise object pour forcer l'utilisation de la visualisatrice SVG même pour du SVG créé à la volée en texte mêlé au XHTML

230

### Exemple de génération multi-formats :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" version="1.0" encoding="iso-8859-1" doctype-public="-//W3C//DTD
XHTML 1.0 Transitional//EN" doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"/>
<xsl:template match="/">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:svg="http://www.w3.org/2000/svg"
xml:lang="fr">
<head>
<title>GENERATION DE SVG ET XHTML EN XSLT</title>
<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
<object id="AdobeSVG" classid="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"></object>
<xsl:processing-instruction name="import">
namespace="svg" implementation="#AdobeSVG"
</xsl:processing-instruction>
</head>
<body>
<h1>Chiffre d'affaires par livre</h1>
<svg:svg width="500" height="500" viewBox="0 0 500 500" version="1.1">
<svg:title> SVG + XSL </svg:title>
<svg:desc> Ceci est un exemple de SVG de base </svg:desc>
<xsl:for-each select="bibliotheque/livre">
<svg:rect x="100px" y="{20*position()}px" width="{@ca}" height="20" style="fill:red; stroke:black;
stroke-width:1">
<svg:animate attributeName="width" attributeType="XML" begin="0s" dur="5s" fill="freeze"
from="0" to="{@ca}"/>
</svg:rect>
<svg:text x="10px" y="{20*position() + 16}px" font-family="Arial" font-size="16px" font-
weight="bold"><xsl:value-of select="titre/text()"/></svg:text>
</xsl:for-each>
</svg:svg> </body></html></xsl:template></xsl:stylesheet>
```

## Conclusion

- ❑ Dans cette partie vous avez vu comment mettre en œuvre des SVG statique pour faire des dessins ou des graphiques dans vos PDF ou vos HTML:

- ✓ qu'est ce que SVG?
- ✓ Sa structure
- ✓ ses différentes formes natives
- ✓ différentes balises
  - Groupes, Texte, liens, symboles, images
- ✓ Des effets graphiques
- ✓ Des animations
- ✓ Les Multi formats et CSS