Formation JavaScript

Perfectionnement et Programmation Orientée Objet

Auteur : Cyrille Tuzi

Plan de la formation

1. Compatibilité et bonnes pratiques

2. Les secrets des données
3. Les secrets des boucles
4. Les secrets des fonctions

5. Programmation Orientée Objet

Compatibilité

Standards et doctype

- Standards CSS : depuis Internet Explorer 8
- **Standards JS : depuis Internet Explorer 9**
- Le seul doctype à utiliser :

```
<!DOCTYPE html>
```

Le cas Internet Explorer

- Internet Explorer 6 et 7 : **< 1%** d'usage
  - http://gs.statcounter.com/

- Windows XP IE (6-) **8**
  - pays en développement, grands comptes, service public
- Windows Vista IE (7-) 9
- Windows 7 IE (8-) **11**
- Windows 8 IE (10-) **11**

- **Pas de vieux IE sur mobiles et tablettes**

Manuels

- WebPlatform
  - http://docs.webplatform.org
- Mozilla Developer Network (MDN)
  - https://developer.mozilla.org/
- W3Schools
  - http://www.w3schools.com/

Bonnes pratiques

JavaScript disponible ?

- JavaScript n'est pas toujours disponible :
  - accessibilité
  - référencement
  - chargement long, voire qui échoue
  - erreurs non prévues

- Les fonctionnalités importantes d'un site doivent donc être opérationnelles sans JavaScript, qui doit seulement être une couche facultative.

Sécurité

- JavaScript se trouve côté client, et n'est donc pas sécurisé :
  - les contrôles de sécurité doivent être faits impérativement côté serveur (côté client, c'est seulement pour l'ergonomie)

Inclusion

- Pour inclure un script :

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

- Mauvaises pratiques :

```
<script>function myFunction() {}</script>
<a onclick="myFunction()">Lien</a>
```

Chargement et performances

- Pour un chargement rapide de la page, en production :
  - inclure **le moins de fichiers possible** (CSS, JavaScript, images, etc.)
  - **minifier** les feuilles de style et les scripts

  - sauf exceptions, **inclure le(s) script(s) en bas de page**, à la fin du corps de la page (*body*), et non dans la partie préliminaire (*head*)

Mode strict

```
'use strict';
```

- Améliore :
  - le debug, et donc, la fiabilité du code
  - les performances
  - la compatibilité avec les futures versions de JavaScript
- Principal changement :
  - déclaration des variables obligatoire

JSDoc

@type {Type}
@enum

@param {Type} varName Description
@param {Type=} varName Description (optional)
@param {...Type}

@return {Type} Description

@deprecated

JSDoc : les types

{bool} {number} {string} {function} {undefined}
{(bool|number)} {*}

{Array} {Object} {Class}
{Array.<number>}
{{property1: number, property2: string}}

{?number} Type simple ou null
{!Class} Objet non null

Style de codage

- HTML et CSS
  - http://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml
- JavaScript
  - http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml

Validateurs et templates

- Validateurs
  - HTML : http://validator.w3.org/
  - CSS : http://jigsaw.w3.org/css-validator/

- Templates de démarrage propres
  - HTML : http://html5boilerplate.com/
  - CSS : http://necolas.github.com/normalize.css/

Validateurs JavaScript

- http://www.jslint.com/
- http://www.jshint.com/

- Vérifications :
  - mode strict et conventions de nommage obligatoires
  - déclaration des variables obligatoire
  - points virgules et accolades obligatoires
  - itérations de propriétés filtrées
  - comparaisons strictes uniquement
  - incrémentations ++ / -- interdites
  - eval() interdit

Console d'erreurs

```
console
    .log("Message générique");
    .info("Message d'information");
    .warn("Message d'avertissements");
    .error("Message d'erreur");
    .dir(dataObject);

    .time('some action');
    .timeEnd('some action');
```

Debug

- La plupart des navigateurs disposent d'un debugger JavaScript intégré.
- Dans Firefox :
  - (> Outils) > Développement web > Debugger

Les secrets des données

Les variables

- Déclaration multiple :

```
var data1 = 1, data2 = 2;
```

- Déclaration obligatoire en mode strict.

- Constantes (ES6) :

```
const MA_VARIABLE = 2;
```

Les nombres

```
var dataNumber = -10.5;
```

- Valeurs particulières :
  - NaN (Not A Number)
  - Infinity / -Infinity

- Éviter les opérations sur les décimaux (problèmes de précision).

Incrémentation

- Préfixe :
  - ++dataNumber;
  - --dataNumber;

- Postfixe :
  - dataNumber++;
  - dataNumber--;

Bases

- Octal (base 8) :
  - dataNumber = 07;
  - Ne pas utiliser, supprimé en mode strict

- Hexadécimal (base 16) :
  - dataNumber = 0xFF;

Les chaînes de caractères

```
var dataString1 = "J'écris";
var dataString2 = '<p id="title">Titre</p>';
```

- Sauts de ligne non autorisés.

Booléens et valeurs spéciales

```
var dataBoolean1 = true;
var dataBoolean2 = false;

var dataNull = null;

undefined;
```

- est une variable, et peut donc être modifié !

Listes automatiques : tableaux

- Tableaux : index automatiques et numériques

```
var dataArray = ['valeur 1', 'valeur 2'];
```

- Usage : `dataArray[0];`
- Ajout : `dataArray.push('valeur 3');`
- Taille : `dataArray.length;`

Gestion des tableaux

dataArray

.push("valeur")	Ajouter une valeur à la fin
.pop()	Retirer la dernière valeur
.unshift("valeur")	Ajouter une valeur au début
.shift()	Retirer la première valeur
.sort()	Trier par ordre alphabétique
.reverse()	Inverser l'ordre

.concat(array2)

.slice(begin, end)

.splice(index, number, newElements...)

Gestion moderne des tableaux

```
dataArray
    .isArray();

    .indexOf("valeur");
    .lastIndexOf("valeur");

    .every(function () {});
    .some(function () {});
    .forEach(function (value, index, array) {});
    .map(function (value, index, array) {});
    .filter(function (value, index, array) {});
```

Chaîne < > tableau

- Transformer un texte en tableau :

```
dataString.split(',');
```

- Transformer un tableau en chaîne :

```
dataArray.join(',');
dataArray.toString();
```

Listes nommées : objets

```
var dataObject = {
    propriete1: 'valeur 1',
    propriete2: 'valeur 2'
};
```

- Accès :
  - statique : dataObject.propriete1;
  - dynamique : dataObject[maVariable];
  - suppression : delete dataObject.propriete1;

JSON

- **J**ava**S**cript **O**bject **N**otation

```
{
    "propriete1": "valeur 1",
    "propriete2": "valeur 2"
}
```

  - Guillemets doubles impératifs pour les propriétés et les valeurs, pas de méthodes.
  - Aussi rigoureux que du XML, ne pas éditer à la main.

Namespace

- Un objet pouvant contenir à la fois des variables et des fonctions, on peut s'en servir pour créer des namespaces.

```
var MonEspace = {

};
```

Chaîne < > objet

- Transformer un texte en objet :

```
JSON.parse('{}');
```

- Transformer un objet en chaîne :

```
JSON.stringify({});
```

Objet ou tableau ?

- Les propriétés d'un objet peuvent aussi être numériques, mais cela reste un objet :

```
{
    0: "valeur 0",
    propriete1: "valeur 1"
}
```

- A éviter, mais c'est le cas de certains objets natifs (comme les listes DOM et arguments).
- Fonctions liées aux tableaux indisponibles.

C'est qui ce type ?

- Pour tester le type d'une variable :
  - typeof maVariable;
  - jQuery (plus précis) : $.type();
- Attention :
  - typeof null == 'object';
  - typeof [1, 2] == 'object';
- Pour les tableaux :
  - [1, 2] instanceof Array

Conversions

- Conversion explicite
- Conversion implicite
  - opérations particulières : dicté par l'opération
  - comparaisons : règles de transtypage

Conversion explicite

- Forcer le type d'une variable :
  - Vers un booléen :

```
!!data;
Boolean(data);
```

  - Vers un nombre :

```
+data;
-data;
Number(data);
```

  - Vers une chaîne de caractères :

```
String(data);
```

Conversion implicite

- Attention à l'addition et la concaténation qui utilisent le même opérateur.

- La concaténation est prioritaire, utiliser des parenthèses si nécessaire.

```
5 + " commentaires"; // 5 commentaires
"Il y a " + (5 + 3) + " commentaires"; // 53
"Il y a " + (5 + 3) + " commentaires"; // 8
```

Vers le booléen

- 5 / Infinity > true
- 0 / -0 / NaN > false

- 'chaîne' > true
- '' > false

- {} / [] > true

- undefined / null > false

Vers le nombre

- false > 0
- true > 1

- " / ' ' > 0
- 'chaîne' > NaN
- '10' > 10

- {} / [1, 2] > NaN
- [] > 0
- [10] > 10

- undefined > NaN
- null > 0

Vers la chaîne de caractères

• false	>	'false'
• true	>	'true'
• 5	>	'5'
• Infinity	>	'Infinity'
• NaN	>	'NaN'
• {}	>	'[object Class]'
• ['val1', 'val2']	>	'val1,val2'
• undefined	>	'undefined'
• null	>	'null'

Comparaisons simples

1. Vérification du type, si identique
   a. undefined == undefined
   b. null == null
   c. Number : NaN != NaN, 0 == -0
   d. Objets si identiques seulement : {} != {}, [] != []
2. null == undefined
3. Number vs String : String => Number
4. Boolean => Number, 2 != true

Copie / référence

- Les valeurs sont passées :
  - par copie pour les types scalaires
  - par référence pour les objets (tableaux compris)

- Cela vaut aussi pour les paramètres de fonctions.

Les secrets des boucles

Boucles et conditions

- Les différentes boucles :
  - `while () {}`
  - `do {} while ()`
  - `for (var i = 0; i < 10; i += 1) {}`

- Les différentes conditions :
  - `if () {} else if () {} else {}`
  - `switch () { case 'valeur': break; default: break;}`
    - Il s'agit de comparaisons strictes

Comparaisons

- Comparaisons simples des valeurs :
  - == / !=
  - Transtypage
- Comparaisons strictes des types et valeurs :
  - === / !==

Tableaux : itération

```
var dataArray = ['valeur 1', 'valeur 2'];

for (var i = 0; i < dataArray.length; i += 1) {

    dataArray[0];

}
```

Objets : itération

```
var dataObject = {
    propriete1: 'valeur 1',
    propriete2: 'valeur 2'
};

for (var propriete in dataObject) {
    dataObject[propriete];
}
```

Problèmes d'itération

- Par défaut, tout ce que contient l'objet est itéré. Pour l'éviter :

```
for (var propriete in dataObject) {
    if (dataObject.hasOwnProperty(propriete)) {
        dataObject[propriete];
    }
}
```

Problèmes d'itération

- **for** (var index **in** obj) {}
  - ne filtre pas les propriétés des prototypes (ex: item dans NodeList)
  - filtre les propriétés natives (ex: length pour les tableaux)

- **propertyIsEnumerable()** / Object**.keys**
  - filtre les propriétés des prototypes
  - filtre les propriétés natives

- **hasOwnProperty() /** Object**.getOwnPropertyNames**
  - filtre les propriétés des prototypes
  - ne filtre pas les propriétés natives

- 'prop' **in** obj
  - ne filtre rien

Les secrets des fonctions

Portée des variables

```
var maVariable = "variable globale";

function maFonction() {

    var maVariable = "variable locale";

}
```

Paramètres facultatifs

- Les paramètres sont toujours facultatifs par défaut. Prévoir des valeurs par défaut.

```
function maFonction(param) {
    if (null == param) {
        param = "valeur par défaut";
    }
}
```

Paramètres obligatoires

```
function maFonction(param) {

    if (null == param) {
        throw "First argument missing";
    }

}
```

Paramètres typés

```
function maFonction(param) {

    if ('string' != typeof param) {
        throw "First argument missing";
    }

}
```

Nombre de paramètres indéfini

```
function maFonction() {
    for (var i = 0; i < arguments.length; i += 1)
{
        arguments[i];
    }
}

maFonction.length; // Arguments attendus
```

Gestion intelligente des paramètres

```
function maFonction(options) {
    var defaults = {
        option1: "valeur par défaut 1",
        option2: "valeur par défaut 2"
    };
    var settings = jQuery.extend({}, defaults, options);
}

maFonction({
    option2: "valeur manuelle"
});
```

Fonction anonyme

- 1ère étape : déclaration

```
var maFonction = function() {
    // Traitements
};
```

- 2e étape : exécution (appel)

```
maFonction();
```

Appels indirects

- Array.prototype.forEach.**call**(elementsList, function() {});

- .apply() pour paramètres multiples sous forme de tableau.

Closure

```
var elements = document.getElementsByTagName('a');

for (var i = 0; i < elements.length; i += 1) {
        elements[i].addEventListener('click', function() {
                alert(i); // Affichera toujours le total d'éléments
        });

        elements[i].addEventListener('click', (function(i) {
                return function() {
                        alert(i); // Affichera la position de l'élément
                };
        })(i));
}
```

Encapsulation

```
(function(window, undefined) {

    'use strict';

    // Traitements

})(this);
```

Programmation Orientée Objet

JSDoc pour la POO

@constructor

@private
@protected
@public

@this {Type}

@extends {Class}
@override

@interface
@implements {Class}

Fonction constructeur

```
var Class = function(propertyValue) {
    this.property = propertyValue;
};

var dataObject1 = new Class('valeur 1');
var dataObject2 = new Class('valeur 2');

dataObject1.property;
```

Prototype

```
Class.prototype = {
    constructor: Class,
    method1: function() {
        this.property;
    },
    method2: function() {}
};

dataObject1.method1();
dataObject1.method2();
```

Objets natifs

- Tout en JavaScript est un objet, y compris les types de données simples :
  - Boolean, Number, String
  - Array, Object, Function
  - Math, Date, RegExp
  - Node, NodeList, Element, HTMLElement,... (IE9+)

- avec son prototype, extensible !
  - Attention à ne pas écraser les méthodes déjà existantes

Problèmes de contexte

```
Class.prototype = {
    method1: function () {},
    method2: function () {
        element.addEventListener('click', function () {
            this.method1(); // Erreur
        });
        setTimeout(function () {
            this.method1(); // Erreur
        }, 5000);
    }
};
```

Garder le contexte

```
Class.prototype = {
    method1: function () {},
    method2: function () {
        element.addEventListener('click', (function () {
            this.method1(); // Erreur
        }).bind(this));
        setTimeout((function () {
            this.method1(); // Erreur
        }).bind(this), 5000);
    }
};
```

Visibilité des propriétés

```
var Class = function() {

    this.property = 'public';

    var localVariable = 'private';

};
```

Visibilité des méthodes

```
var Class = function() {
    this.privilegedMethod = function() {};
    var privateMethod = function() {
        // Attention à this, le contexte a changé
    };
};

Class.prototype.publicMethod = function() {};
```

Héritage du constructeur

```
var ParentClass = function(prop1, prop2) {};

var ChildClass = function(prop1, prop2) {
    ParentClass.call(this, prop1, prop2);
    /* ou */
    ParentClass.apply(this, [prop1, prop2]);
};
```

- Héritage multiple possible

Héritage du prototype

```
var ParentClass = function() {};
ParentClass.prototype = {
    constructor: ParentClass
};

var ChildClass = function() {};
ChildClass.prototype = new ParentClass ();
ChildClass.prototype.constructor = ChildClass;
ChildClass.prototype.newMethod = function()
{};
```

Tester la classe

```
var monObjet = new ChildClass();

monObjet instanceof ChildClass;
// true

monObjet instanceof ParentClass;
// true (seulement dans le cas d'un héritage du prototype)
```

Copie manuelle du prototype

```
var ParentClass = function() {};
var ChildClass = function() {};

for (var propriete in ParentClass.prototype) {
    ChildClass.prototype[propriete] =
        ParentClass.prototype[propriete];
}
ChildClass.prototype.constructor = ChildClass;
```

Copie manuelle via jQuery

```
var ParentClass = function() {};
var ChildClass = function() {};

jQuery.extend(ChildClass.prototype,
        ParentClass.prototype);
```

Appel parent

```
var ParentClass = function() {};

ParentClass.prototype.method = function() {};

var ChildClass = function() {};

ChildClass.prototype.method = function() {
    ParentClass.prototype.method.call(this);
};
```

Méthodes et propriétés statiques

```
var Class = function() {};

Class.staticProperty = 'value';

Class.staticMethod = function() {};
```

Classes et méthodes abstraites

```
var AbstractClass = function() {
    throw "Instanciation non autorisée";
};

Class.prototype.abstractMethod = function() {
    throw "Méthode non définie";
};
```

Nouvelles fonctions

```
Object.
    create(prototype);

    getPrototypeOf(object);
    keys(object);                          // Enumérables
    getOwnPropertyNames(object);    // Toutes

    preventExtensions(object); / isExtensible(object);
    seal(object);                        / isSealed(object);
    freeze(object);                      / isFrozen(object);
```

Config des propriétés

```
Object.
    create(prototype, properties);
    defineProperty(objet, propName, description);
    defineProperties(objet, {
        propName1: description1,
        propName2: description2
    });
```

- Paramètres généraux :
  - configurable: false | true
  - enumerable: false | true

Config des propriétés

- Constantes :
  - value: 'valeur'

- Propriétés publiques :
  - value: 'valeur'
  - writable: true

- Propriétés privées:
  - get: function () { return localVariable; }
  - set: function (newValue) { localVariable = newValue; }