

# WEB Engineering: JavaScript



JavaScript

Press Space for next page →



# JavaScript

- Définition
- Syntaxe
- Les objets
- Les évènements
- Accès à un élément quelconque d'une page

# JavaScript

- Mais à quoi ça sert ?
  - Manipulation et animation dynamiques du contenu des pages Web
  - Communication avec « le » serveur
    - Warning : cross-domain scripting
  - Enregistrement des actions de l'utilisateur (!)
  - Réaction aux évènements utilisateur
  - Sauvegarde de données...

JavaScript est un langage de programmation orienté objet développé par NETSCAPE dans les années 90, et s'appelait à l'origine LiveScript.

- Historique:
  - Il fut adopté par la firme SUN (qui est à l'origine du langage JAVA) en 1995.
  - Le JavaScript est une extension du langage HTML.

# Popularité de JavaScript

- JavaScript présent dans tous les navigateurs web avec maintenant des implémentations à jour
- ECMAScript 6 disponible et implémentation dans les différents moteurs
- Node.js et NPM à la base du nouveau succès de JavaScript (après avoir été le langage le plus détesté)
- Le plus de modules tiers disponibles grâce à NPM et GitHub. Il existe un module JavaScript existant pour quasiment chaque besoin, ou bien il est en train d'être écrit !

# Ubiquité

- On peut utiliser JavaScript :
  - Sur navigateur pour sites web et web apps
  - Sur serveur pour applications web
  - Sur navigateur + serveur pour applications isomorphiques
  - Sur serveur et poste développeur pour scripts/batches systèmes (export CSV, transformation/conversion de données, etc.)
  - Sur serveur et poste développeur pour scripts web (SlimerJS, PhantomJS, etc.)

# JavaScript caractéristiques

- Orienté “prototype” ???
  - Basé objet mais pas orienté objet
  - On déclare un objet générique, puis héritage
- Dynamique ???
  - Typage, fonctions, code...
  - Pratique mais dangereux...
- Événementiel ???
  - Paradigme de programmation
  - Attente” puis réaction aux actions “utilisateur”

# JavaScript

- Définition
- **Syntaxe**
- Les objets
- Les évènements
- Accès à un élément quelconque d'une page

# Syntaxe

- Intégration du script dans la page

```
*inline*
<script type="text/javascript">....</script>
```

Appel externe avec une réf. comme pour css

```
<script src="/js/script.js" type="text/javascript"/>
```

- Chargement du script

- Importance du placement dans la page
- Dans le header (délai)
- Dans le body, ou en fin de body
- Cache si appel externe (mieux), compression

# Syntaxe

- Bonnes pratiques
  - Fichier .js externe
- Lecture par événement “onload”
- “Separation of concerns” (Dijkstra)
  - Pour le script principal
  - Utilisation de fonctions
- chargement dynamique (cf plus tard)

# Syntaxe

- Le code JavaScript se trouve dans un fichier ayant pour extension .js
- L'inclusion de ce fichier se fait généralement au début des pages HTML grâce à la balise script:

```
<script type="text/javascript" src="url/nomdufichier.js"> </script>
```

- Les structures conditionnelles ou répétitives (boucles)ont la même syntaxe qu'en langage C.
- JavaScript est un langage faiblement typé.
- Portée des variables:
  - Une variable déclarée en début de script sera considérée comme étant globale.
  - Les variables utilisées/déclarées dans une fonction restent locales à la fonction.

# Syntaxe

- The JavaScript syntax is similar to C# and Java
  - Operators (\*+, =, !=, &&, ++,...)
  - Variables (typeless)
  - Conditional statements (**if, else**)
  - Loops (**for, while**)
  - Arrays (**my\_array**) and associative arrays (**my\_array'abc'**)
  - Functions (can return value)
  - Function variables (like the C# delegates)

# JavaScript data types

- Numbers (integer, floating-point)
- Boolean (true / false)
- String type – string of characters

```
var myName = "You can use both single  
or double quotes for strings";
```

- Arrays

```
var my_array = [1, 5.3, "aaa"];
```

- Associative arrays (hash tables)

```
var my_hash = {a:2, b:3, c:"text"};
```

# Everything is Object

- Every variable can be considered as object
- For example strings and arrays have member functions:

```
// file: objects.html
var test = "some string";
alert(test[7]);           // shows letter 'r'
alert(test.charAt(5));    // shows letter 's'
alert("test".charAt(1));  // shows letter 'e'
alert("test".substring(1,3)); // shows 'es'
var arr = [1,3,4];
alert (arr.length); // shows 3
arr.push(7);           // appends 7 to end of array
alert (arr[3]);         // shows 7
```

# String Operations

The + operator joins strings (slow)

```
string1 = "lol ";
string2 = "cats";
alert(string1 + string2); // lol cats
```

What is "11" + 11?

```
alert("11" + 11); // 1111
```

Converting string to number:

```
alert(parseInt("11") + 11); // 22
alert(parseInt("011",8) + 11); // 20 ???
alert(parseInt("011", 10) + 11); // 22
```

# Arrays Operations and Properties

Declaring new empty array:

```
var arr1 = new Array(); var arr2 = [];
```

Declaring an array holding few elements:

```
var arr = [1, 2, 3, 4, 5];
```

Appending an element / getting the last element:

```
arr.push(3); var element = arr.pop();
```

Reading the number of elements (array length):

```
arr.length;
```

Finding element's index in the array:

```
arr.indexOf(1);
```

# Conditional Statement (if)

```
unitPrice = 1.30;
if (quantity > 100) {
    unitPrice = 1.20;
}

/*
Greater than: >
Less than: <
Greater than or equal to: >=
Less than or equal to: <=
Equal: ==
Not equal: !=
*/
```

# Conditional Statement (if)

The condition may be of Boolean or integer type:

```
var a = 0;
var b = true;
if (typeof(a)=="undefined" || typeof(b)=="undefined") {
    document.write("Variable a or b is undefined.");
}
else if (!a && b) {
    document.write("a==0; b==true;");
} else {
    document.write("a==" + a + "; b==" + b + ";");
}
```

# Switch Statement

The switch statement works like in C#:

```
switch (variable) {  
    case 1:  
        // do something  
        break;  
    case 'a':  
        // do something else  
        break;  
    case 3.14:  
        // another code  
        break;  
    default:  
        // something completely different  
}
```

# Loops

Like in C# (for, while, do while)

```
/*
for loop
while loop
do ... while loop
*/
var counter;
for (counter=0; counter<4; counter++) {
    alert(counter);
}
for (var key in haystack);
while (counter < 5) {
    alert(++counter);
}
```

# Syntaxe

- Les fonctions:
  - La syntaxe de définition d'une fonction est:

```
function nomFonction(arg1,arg2,...)
```

- Contrairement au langage C, on ne donne pas le type des arguments ni celui de la valeur de retour éventuelle

# Syntaxe

- Les chaînes de caractères:
- Une chaîne de caractères est délimitée par des guillemets simples ou doubles.
- Exemple:

```
texte = "un beau texte"
```

Ou

```
texte = 'un beau texte'
```

# Syntaxe

- Les chaînes de caractères:
  - Pour manipuler des chaînes de caractères, il existe de nombreuses fonctions.
  - Contrairement au langage C, il est possible de comparer deux chaînes de caractères à l'aide de l'opérateur ==.

# Functions

- Code structure – splitting code into parts
- Data comes in, processed, result returned

```
function average(a, b, c) {  
    var total;  
    total = a+b+c;  
    return total/3;  
}
```

- Anonymous functions

```
var average = function (a, b, c) { ... }  
var res1 = average(1,2,3);  
var myAvg = average;  
var res2 = myAvg(1,2,3);
```

# Function Arguments and Return Value

- Functions are not required to return a value
- When calling function it is not obligatory to specify all of its arguments
- The function has access to all the arguments passed via arguments array

```
function sum() {  
    var sum = 0;  
    for (var i = 0; i < arguments.length; i++)  
        sum += parseInt(arguments[i]);  
    return sum;  
}  
alert(sum(1, 2, 4));
```

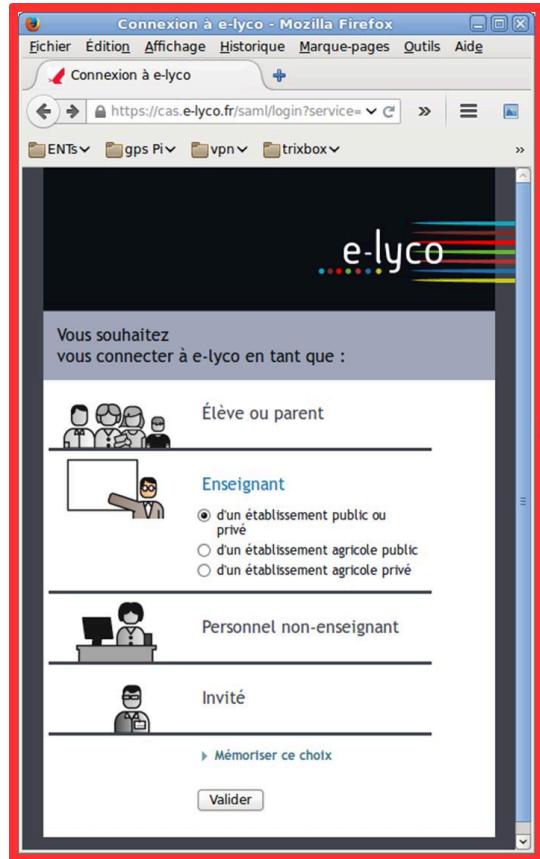
# JavaScript

- Définition
- Syntaxe
- **Les objets**
- Les évènements
- Accès à un élément quelconque d'une page

# Les objets

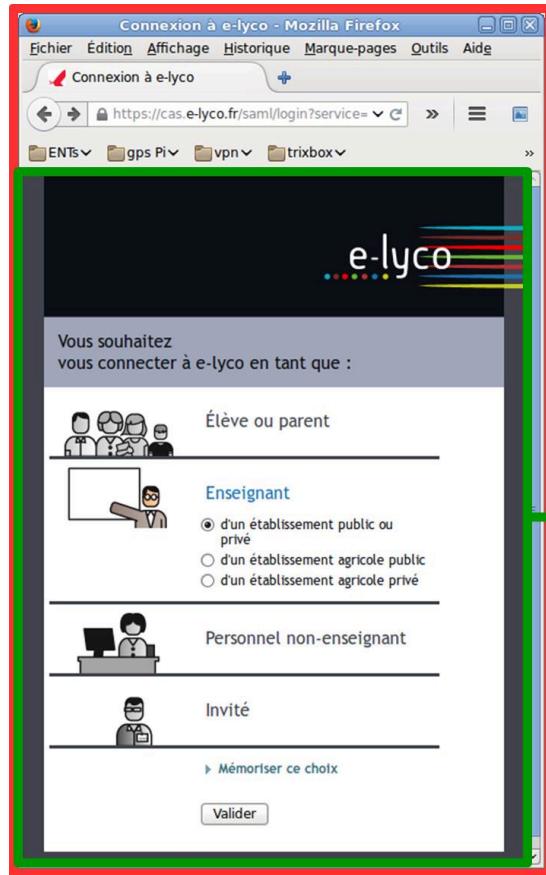
- L'utilisation principale de JavaScript est la manipulation des objets d'une page et plus particulièrement des objets des formulaires
- Une page est composée de façon hiérarchique.

# Les objets



Objet  
fenêtre

# Les objets



# Les objets

Connexion à e-lyco - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils Aide

Connexion à e-lyco

Vous souhaitez vous connecter à e-lyco en tant que :

Élève ou parent

Enseignant

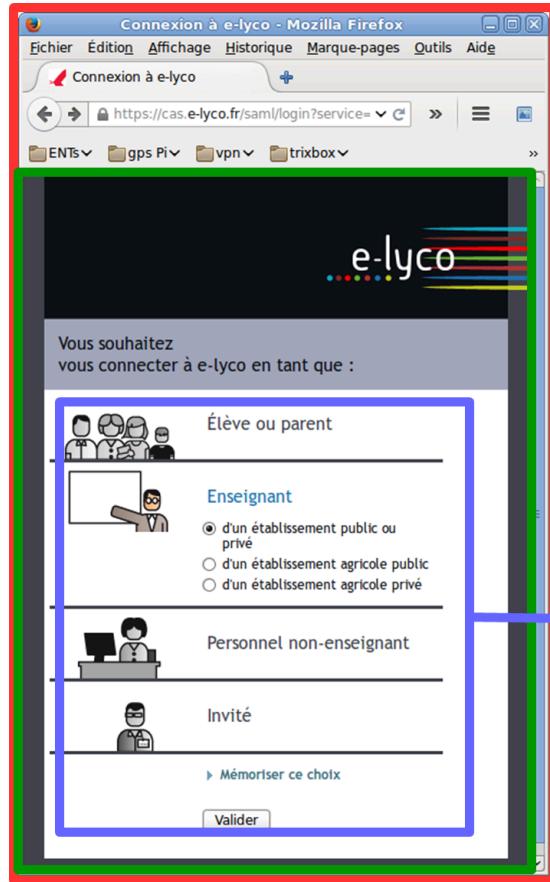
dun établissement public ou privé  
 dun établissement agricole public  
 dun établissement agricole privé

Personnel non-enseignant

Invité

Mémoriser ce choix

Valider



Objet formulaire

# Les objets



Objets radio

Objet bouton

# Les objets

- L'accès se fait de façon hiérarchique.

```
window.document.forms["nomDuformulaire"].nomDeLélément
```

- Pour chaque niveau il existe des méthodes et des attributs. Exemple:

```
<html><body>
  <form name="monForm">
    <label for="login">Votre login :</label>
    <input type="text" name="login" id="login" />
  </form>
</body></html>
```

- Si je souhaite avoir la valeur contenue dans le champ login du formulaire, j'utiliserai le code JavaScript suivant:

```
leLogin=window.document.forms["monForm"].login.value;
```

# Les objets

- Classes are functions
- Objects are associative arrays
- **this** refers to the owning code
- **new** copies the prototype into new reference

```
var person = new Object();
person.first = "Tim";
person.last = "Scarfe";

person.sayHello = function() {
    console.log("Hello, " + this.first + " " + this.last);
};

person.sayHello();
```

# Les objets

```
function User(first, last) {
    this.first = first;
    this.last = last;

    this.sayHello = function() {
        console.log("Hello, " + this.first
                    + " " + this.last);
    }
}

var svi = new User("Svi", "Ivanov");
var alex = new User("Alex", "Ivanova");
svi.sayHello();
alex.sayHello();
```

# JavaScript

- Définition
- Syntaxe
- Les objets
- **Les évènements**
- Accès à un élément quelconque d'une page

# Les évènements

- L'action sur un élément de la page se fait lors d'un événement particulier (clique de souris, changement de la valeur d'un champ, etc)
- Voici quelques évènements possibles pour une page WEB
  - Click (*onClick*)
  - Load (*onLoad*)
  - Unload (*onUnload*)
  - MouseOver (*onMouseOver*)
  - MouseOut (*onMouseOut*)
  - Focus (*onFocus*)
  - Change (*onChange*)
  - Submit (*onSubmit*)

# Les évènements

- Pour qu'un objet réagisse à un événement, il faut lui associer une fonction de traitement.
- Exemple:

```
<script src="mesFonctions.js" type="text/javascript"></script>
<form name="monForm">
    <label for="login">Votre login :</label>
    <input type="text" name="login" id="login"
        onchange="afficheLogin();">
    <input type="text" name="loginBis" id="loginBis">
</form>
```

Result    JavaScript    HTML    CSS



Edit in JSFiddle

Votre login :

- Le fichier mesFonctions.js contient le code suivant:

```
function afficheLogin(){
    window.document.forms["monForm"].loginBis.value =
        window.document.forms["monForm"].login.value;
}
```

# JavaScript

- Définition
- Syntaxe
- Les objets
- Les évènements
- **Accès à un élément quelconque d'une page**

# Accès à un élément quelconque d'une page

- La manière la plus simple pour avoir accès à un élément quelconque d'une page est d'utiliser son identifiant

```
var obj = document.getElementById("idelement") ;
```

- Il est ainsi possible d'avoir des informations et d'agir sur l'élément
  - **innerHTML** : va récupérer/modifier le contenu HTML de l'élément
  - **textContent** : va récupérer/modifier le contenu de l'élément
  - **nodeName** : va récupérer le nom de l'élément

# An example

```
<div id="div1"></div>
<input type="button" value="Vert" onclick="vert();"/>
<input type="button" value="Rouge" onclick="rouge();"/>

#div1{
    background-color:#ff0000;
    width:200px;
    height:200px;
}

function vert(){
    document.getElementById('div1').style.backgroundColor='#00ffff';
}
function rouge(){
    document.getElementById('div1').style.backgroundColor='#ff0000';
}
```

Result

JavaScript

HTML



Edit in JSFiddle

CSS

Vert Rouge

# JSON

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent \*
- Key-value pairs

# JSON

- JSON uses JavaScript syntax, but the JSON format is text only.
- Text can be read and used as a data format by any programming language.
- The JSON format was originally specified by Douglas Crockford.

```
{  
    "id": 1,  
    "name": "A green door",  
    "price": 12.50,  
    "tags": ["home", "green"]  
}
```

# EcmaScript 2015, es6 : Let and Const Declarations

ES6 introduces two new ways to declare variables: let and const. The let keyword allows for block-scoped variables, reducing the risk of variable hoisting issues. const, on the other hand, is used to define constants, preventing accidental reassignment of values. These features bring more predictability and maintainability to your code.

	var	let	const
Stocké en global ?	✓	✗	✗
Se limite à la portée d'une fonction ?	✓	✓	✓
Se limite à la portée d'un block ?	✗	✓	✓
Peut être réassigné ?	✓	✓	✗
Peut être redéclaré ?	✓	✗	✗

# EcmaScript 2015, es6 : Arrow Functions

Arrow functions provide a more concise syntax for writing anonymous functions. They use the => operator and automatically capture the value of this from the surrounding context. This simplifies function definitions, especially for short, one-liner functions.

```
const materials = ['Hydrogen', 'Helium', 'Lithium', 'Beryllium'];

console.log(materials.map((material) => material.length));
// Expected output: Array [8, 6, 7, 9]
```

# EcmaScript 2015, es6 : Template Literals

Template literals offer a cleaner way to handle string interpolation in JavaScript. They use backticks (`) to enclose strings, allowing variables to be embedded directly within the string using \${variable}. This feature simplifies string concatenation and enhances code readability.

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}! <BR>
How are you
`;
```

# EcmaScript 2015, es6 : Destructuring Assignment

Destructuring enables you to extract values from arrays and objects concisely. It reduces the need for manual property access, making your code more efficient and readable.

```
let a, b, rest;  
[a, b] = [10, 20];  
  
console.log(a);  
// Expected output: 10  
  
console.log(b);  
// Expected output: 20  
  
[a, b, ...rest] = [10, 20, 30, 40, 50];  
  
console.log(rest);  
// Expected output: Array [30, 40, 50]
```

# EcmaScript 2015, es6 : Spread and Rest Operators

The spread (...) and rest operators are incredibly versatile. They allow you to spread the elements of an array into individual variables or combine variables into an array. These features are particularly useful in functions that accept a variable number of arguments.

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const combined = [...arr1, ...arr2];
console.log("Combined array:", combined); // [1, 2, 3, 4, 5, 6]
```

# EcmaScript 2015, es6 : Classes

ES6 introduces a class syntax for creating objects, making it more structured and similar to other object-oriented languages. It simplifies the process of defining constructor functions and prototypes, leading to more organized and maintainable code.

```
class Rectangle {  
    constructor(hauteur, largeur) {  
        this.hauteur = hauteur;  
        this.largeur = largeur;  
    }  
  
    get area() {  
        return this.calcArea();  
    }  
  
    calcArea() {  
        return this.largeur * this.hauteur;  
    }  
}  
  
const carre = new Rectangle(10, 10);  
console.log(carre.area());
```

# EcmaScript 2015, es6 : Promises

Promises are a powerful tool for handling asynchronous operations in a more readable and manageable way. They simplify error handling and help avoid callback hell, making code that relies on asynchronous data flow more predictable and maintainable.

Une promesse en JavaScript est un objet qui représente l'état d'une opération asynchrone. Une opération asynchrone peut être dans l'un des états suivants :

- Opération en cours (non terminée) ;
- Opération terminée avec succès (promesse résolue) ;
- Opération terminée ou plus exactement stoppée après un échec (promesse rejetée).

```
function loadScript(src){  
    return new Promise((resolve, reject) => {  
        let script = document.createElement('script');  
        script.src = src;  
        document.head.append(script);  
        script.onload = () => resolve('Fichier ' + src + ' bien chargé');  
        script.onerror = () => reject(new Error('Echec de chargement de ' + src));  
    });  
}  
const promesse1 = loadScript('boucle.js');  
const promesse2 = loadScript('script2.js');
```

# EcmaScript 2015, es6 : Modules

ES6 introduces a built-in module system for structuring and organizing code. Modules allow you to split your code into separate files, making it more maintainable and reusable. They provide a clear way to export and import functions and data across different parts of your application.

```
export const name = "square";

export function draw(ctx, length, x, y, color) {
  ctx.fillStyle = color;
  ctx.fillRect(x, y, length, length);
  return {
    length: length,
    x: x,
    y: y,
    color: color,
  };
}
```

- Import features

```
import { name, draw } from "./modules/square.js";
```

# EcmaScript 2015, es6 : Modules

Une méthode plus concise consiste à exporter l'ensemble des valeurs grâce à une seule instruction située à la fin du fichier : les valeurs sont séparées par des virgules et la liste est délimitée entre accolades :

```
export { name, draw } from "./modules/square.js";
```

# EcmaScript 2015, es6 : Default Parameters

With ES6, you can define default values for function parameters. This simplifies function calls and reduces the need for explicit checks for missing arguments.

```
function multiply(a, b = 1) {  
    return a * b;  
}  
  
console.log(multiply(5, 2));  
// Expected output: 10  
  
console.log(multiply(5));  
// Expected output: 5
```

# EcmaScript 2015, es6 : Enhanced Object Literal Syntax

ES6 enhances the object literal syntax by allowing shorthand property and method declarations. This simplifies object creation and results in more concise and readable code.

```
// Variable declaration
var name = "Lilly";
var color = "White";
var age = 3;

// function declaration
// using "this" keyword to access the object keys.
var barkWithName = function(){
    console.log('Woof Woof!!, I am '
    +this.name+' and I am a '
    +this.age+' years old, '
    +this.color+ ' coloured dog.Woof Woof!!');
}

// Using Object Literal Enhancement
// combines all variables into a yetAnotherDog object
var yetAnotherDog = {name, color, age, barkWithName};
yetAnotherDog.barkWithName();
```