JavaScript

Introduction

- **Introduction**
- **Structure**
- **Variables**
- **Fonctions**
- **Expressions régulières**
- **Objets**
- **DOM**
- **BOM**
- **Ajax**

Introduction

- **Langage de script orienté prototype**
- **Interprété par le navigateur et quelques runtimes (NodeJS)**
- **Histoire :**
  - 1995 – Sun/Netscape annonce JavaScript (ancien LiveScript)
  - 1996 – JavaScript dans Netscape 2.0 / JScript dans IE3
  - 1997 – Standard ECMAScript
  - 1998 – Adobe : ActionScript
- **ECMA : Organisme privé européen de standardisation**

Introduction

- **La structure des instructions est héritée du C**
- **Les instructions sint délimitée par ;**
- Les points-virgules sont optionnels
  - Système d'insertion automatique si pas d'ambiguïté et retour chariot
  - Peut amener à des erreur
  - Très fortement conseillé

Introduction

- Booléen (true / false)
- Nombre
  - Entier décimal : `0, 5, -50, 77`
  - Entier octal ou hexadécimal : `077 => 63, 0xA => 10`
  - Réel : `1000.566, .034, 2.75e-2`
- Chaîne : `"", "Hello", 'Kitty'`
- Tableaux : `[], [1, 2, 3], [true, 'hello', 1, {}]`
- Map : `{}, {clef : 'valeur'}, {"clef" : {}, 'clef 2' : []}`

Coercition

- La VM Javascript va tenter de changer automatiquement le type de données pour réaliser les opérations demandées
- Rend possible le mélange de données de différents types
- Une grosse source d'erreur :
  - `[] == false // → true`
  - `null == false // → false`
  - `Number(true) // → 1`
  - `1 == '1' // → true`
- Pour palier à ce problème → `===`
  - `1 === '1' // → false`

Objets standards

- Array, Boolean, Number, Date, String, Regexp, Math, Function

```
var a = new Array(1,2,3); a.sort().reverse();
var b = new Boolean(); b.valueOf();
var n = new Number('5'); n.toLocaleString();
var d = new Date(); d.getTime();
var s = new String('test'); s.toUpperCase();
var r = new RegExp('^.{3}$');
var r = /^.{3}$/;
r.test('abc'); r.exec('abc');
Math.PI; Math.random();
var add = function toto(a, b) {return a + b;};
add.name; // → 'toto'
```

Structuration du code

- Le if

```
if(/* test */) {
   /* liste d'instructions */
} else {
   /* autres instructions */
}
```

- Opérateur ternaire :

```
var a = /* test */ ? /* si true */ : /*
si false */ ;
```

- La valeur du test est convertie en boolean

Structuration du code

- Le switch / case

```
switch (/* variable */) {
case /* valeur 1 */ :
   /* liste d'instructions */
   /* attention au fall through */
case /* valeur 2 */ :
   /* liste d'instructions */
   break;
case /* valeur x */
   /* Liste d'instructions */
   break;
default :
   /* liste d'instructions */
}
```

Structuration du code

- while

```
while(/* test */) {
   /* liste d'instructions */
}
```

- **Do while**

```
do {
   /* liste d'instructions */
} while( /* test */);
```

- **Toujours faire attention à la conversion du test en booléen**

Structuration du code

- for

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

- **for in Object**

```
var obj = {prop1 : 1, prop2 : 2};
for(prop in obj) {
   console.log('prop', prop, '=', obj[prop]);
}
```

- **Attention au for in Array**

```
var arr = [1,2,3,4,5,6];
for(i in arr) {
   console.log('index', i, 'value', arr[i]);
}
```

Structuration du code

Rupture de flux

- Il existe plusieurs solutions pour changer le flux du programme
  - **continue** : continue la boucle avec le prochain élément
  - **break** : sort de la boucle et continue le code
  - **return** : sort de la fonction et continue le code
  - **throw** : sort du traitement classique et remonte un problème de fonctionnement

Les variables

- Il n'y a pas de contrainte de longueur
- Un identifiant JavaScript doit commencer par une lettre ou $ ou _
- Les caractères qui suivent peuvent être également des chiffres
- Il est possible d'utiliser des caractères Unicode (déconseillé)
- Ne doit pas correspondre à un mot réservé

```
var maVariable1;
var maVariable2 = 'ma valeur';
var maVariable3, maVariable4 = 'autre';
```

Les variables

- Les variables ont un typage dynamique

```
var a = 5;
a = 'ma valeur';
a = {clef : 'autre'};
```

- Initialiser une zone mémoire pour une variable avec var
  - En lecture on a un undefined
  - En écriture on définit une variable globale
  - L'utilisation de variables globales est à éviter

```
console.log(b); // → undefined
b = 5;
console.log(b); // → 5
```

Les fonctions

- Une fonction est un ensemble d'instructions
  - Une fonction ***peut*** être appelée avec des arguments
  - Une fonction retourne une valeur
- Définition d'une fonction
  - Le mot clef : function
  - Un nom optionnel
  - Une liste de paramètres entre parenthèse (peut être vide)
  - Un corps entre accolades (peut être vide) ;

```
function nomDelaFonction(arg1, arg2) {
  /* instructions */
}
```

Les fonctions

- Règle de nommage :
  - Les noms de fonctions sont sujettes aux mêmes règles que les noms de variable
  - Une fonction sans nom est dite anonyme
  - La propriété name d'une fonction donne son nom
- Les fonctions sont des objets
  - Il est possible de les affecter à des variables

```
var fn = function nomDeLaFonc(arg1, arg2) {
   /* instructions */
}
```

Les fonctions

- Créer une fonction nommée crée également une référence du même nom

```
// Fonction nommée "foo"
function foo() { /* */ };

// variable pointant sur une fonction existante
var fooVar = foo;

// variable pointant sur une fonction nommée
var barVar = function bar() { /* */ };

// variable pointant sur une fonction anonyme
var bazVar = function() { /* */};

// Noms de fonctions
foo.name === 'foo';
fooVar.name === 'foo';
barVar.name === 'bar';
bazVar.name === '';
```

Les fonctions

- Pour appeler une fonction il faut disposer d'une référence
- L'appel d'une fonction se fait avec les parenthèses
- Il est possible de lui passer autant d'argument que l'on veut

```
var foo = function bar(ag1, arg2) {
  console.log(ar1, arg2);
};

console.log(foo, bar); // → ReferenceError : bar is not defined
foo(); // → undefined, undefined
foo(1, 'coucou'); // → 1 coucou
foo(1, 2, 3, 4,) // → 1 2

// arguments est un mot clef retournant la liste des arguments
foo = function() {
  console.log(arguments);
}

foo(1, 2, 3, 4); // → [1, 2, 3, 4]
```

Les fonctions

Programmation fonctionnelle

- Il est possible de passer une variable contenant une fonction en argument d'une autre fonction
- On peut donc passer un comportement en paramètre

```
function pourChaque(tableau, action) {
  for(index in tableau){
     action(tableau[index]);
  }
}

pourChaque([1, 2, 3, 4], console.log);
```

- La fonction est passée sans parenthèse !!
- Il s'agit d'une référence sur la fonction qui est passée

Portée des variables et des fonctions

- La limite dans laquelle une référence est visible s'appelle un ***scope***
- Visibilité des symboles au sein d'un scope
  - Fonction nommée : utilisable partout dans le scope, même avant sa déclaration (forward-reference)
  - Variable locale : utilisable partout dans le scope, elle vaut undefined avant son initialisation
  - Variable globale : devient une propriété du scope par défaut (window dans un navigateur)
- Les scopes sont délimités par les corps de fonctions
  - Les corps if/for/while ne créent pas de scope
  - c'est différent de la plupart des langages

Portée des variables et des fonctions

```
function scope() {
  // forward reference
  var valeur1 = foo() ;
  function foo() {
      return 42;
  }
  var valeur2 = foo() ;

  // Création des variables
  console.log('avant a', a); // → undefined
  var a = 2;
  console.log('après a', a); // → 2

  // Manipulation des scopes
  if(true) {
     var banane = 'banane';
  }
  console.log(banane); // → banane
}
```

L’isolation et les fonctions anonyme

- Les variables globales posent de nombreux problèmes
- Surtout dans les navigateurs par manque de modularisation
- Solution : la fonction anonyme auto-appelante

```
var a = 'a est une variable globale';

(function(b) {
  var c = 'b et c sont des variables
locales';
}(a)); // a est passé en paramètre
```

Les closures

- Closure signifie fermeture
  - Le principe est de capturer un scope
  - Et le rendre disponible pour une autre fonction
  - Ce principe s'applique à la déclaration d'une fonction
  - Le scope courant est alors capturé

```
var bar = 'value';
function foo() {
   // bar a été capturé et est
   // visible par closure
   console.log(bar);
}
```

Les closures

- Closure : le piège
  - Les closures peuvent sembler très naturelles
  - Elles sont très utilisées en JavaScript
  - Attention aux pièges
  - Il s'agit d'une référence qui est capturée, les données ne sont pas copiées dans la nouvelle fonction

```
var a = ["elem1", "elem2", "elem3", "elem4"];

for(var i = 0; i <3; i++) {
   setTimeout(function() {
   console.log(a[i]);
   }, 1000);
}

// → après 1s on obtient : elem4 elem4 elem4
```

Les expressions régulières

- Chaînes de caractères avec une syntaxe particulière
  - Indique l'occurrence de certains caractères
  - Permet s'extraire certains motifs
- Classe RegExp
  - `var test = new RegExp('Iron');`
  - `var test = new RegExp('iron', 'gi');`
  - `var test = /iron/gi`

Les expressions régulières

- Les méthodes
  - `var reg = new RegExp('iron', 'gi');`
    `reg.test('i am iron man'); → true`
    `reg.test('Iron Maiden'); → true`
    `reg.test('sad but true'); → false`
  - `reg.exec('Black Sabbath plays Iron Man')`
    `→ ["Iron", index: 20, input: "Black Sabbath plays Iron Man"]`
- Méthode de la classe String
  - `'Black Sabbath plays iron man with Iron Maiden'.match(/iron/gi)`
    `→ ["iron", "Iron"]`
  - `'Black Sabbath plays iron man with Iron Maiden'.search(/iron/gi)`
    `→ 20`

Les expressions régulières

- Les méta caractères
  - `var reg = /\?/ ;`
    `var reg  =new RegExp('\\?');` // double échappement
  - . : n'importe quel caractère
  - ^ : commencement
    - `/^iron/.test('iron maiden') → true`
    - `/^maiden/.test('iron maiden') → false`
  - $: fin
    - `/iron$/.test('iron maiden') → false`
    - `/maiden$/.test('iron maiden') → true`
  - * : 0 ou n fois
    - `/a*/.test('aaabcd') → true`
    - `/a*/.test('bcd') → true`

Les expressions régulières

- Les méta caractères
  - ? : 0 ou une fois
    - /a?/.test(‘bcd’) → true
    - /a?/.test(‘aaabcd) → false
  - {n} : exactement n fois
    - /a{2}*/.test(‘aaabcd) → true
    - /a{2}/.test(‘abcd’) → false
  - x{n,} x{n,m} : au moins n fois, entre n et m fois
  - [abc] : a ou b ou c
  - [^abc] : ni a, ni b, ni c
  - \ : échappement
  - a|b : a ou b
  - (abc) : capture de abc
  - x(?=y) x(?!y) : x suivi de y, y si non suivi de y

Les expressions régulières

- Les méta caractères
  - \t → tabulation
  - \n → nouvelle ligne
  - \r → retour chariot
  - \f → saut de page
  - \cX → caractère contrôle, ex : crtl+w
  - \b \B → fin de mot, début de mot
  - \v → tabulation verticale
  - \0 → null
  - \d \D → [0-9], [^0-9]
  - \w \W → [A-Za-z0-9_], [^A-Za-z0-9_]
  - \s \S → espace, tout sauf espace

Les expressions régulières

- **Les groupes**
  - /ironiron/g = /(iron){2}/g
  - /(iron(maiden)?)/ → « iron » ou « iron maiden »
- **Les backreferences**
  - var reNumbers = /#(\d+)/;
    reNumbers.test(“#123456789”);
    RegExp.$1 → ‘123456789’
  - var reMatch = /(\d{4}) (\d{4})/;
    var result = “1234 5678”.replace(reMatch, “$2 $1”);
    → ‘5678 1234’

Les expressions régulières

- **Exemple**

  var reEmail = /^(?:\w+\.?)*\w+@(?:\w+\.)+\w+$/;

  - (?:\w+\.?) : un ou plusieurs mots alphanumériques suivi ou non d'un point
  - \w+@ : un mot alphanumérique suivi de @.
    ((?:\w+\.) : un ou plusieurs mots alphanumériques suivi d'un point.
  - \w+$: doit se finir par un mot alphanumérique.

  ```
  function isValidEmail(sText) {
      var reEmail = /^(?:\w+\.?)*\w+@(?:\w+\.)+\w+$/;
      return reEmail.test(sText);
  }'
  ```

Les objets

- JavaScript est un langage orienté objet à prototype
- Il n'y a pas de notion de classe, seulement d'objet
- Les objets ont une notion de prototype
- Un prototype est aussi un objet
  - Possibilité de chaînage
- Un objet accède de façon transparente à son prototype
- Les objets ont des propriétés de n'importe quel type
- Les propriétés qui ont comme valeur une fonction sont généralement appelées méthodes
- Le nom d'une propriété est appelée clef

Les objets

- Il est possible d'assigner des valeurs aux propriétés
- Mais aussi d'en ajouter et d'en supprimer

```
var o = new Object();

var x = {
  prop0 : 'initial'
};

o.nom = 'test' ;

x.prop1 = function() {
  console.log('hello') ;
} ;

delete x.prop0;

console.log(o); // → { nom : 'test'}
console.log(x); // → { prop1 : function}
```

Les objets : this

- Au sein d'une fonction, on dispose du mot clef *this*
- Les comportement est dépendant de la manière dont la fonction est appelée
  - Appel simple : this = window (navigateur) ou global (NodeJS)
  - Propriété d'un objet : this = l'objet
  - Constructeur (new) : this = l'objet créé
- Il est possible de forcer le contexte (le this)
  - En utilisant call ou apply

```
fn.call(ctxt, arg1, arg2);
fn.apply(ctxt, [arg1, arg2]);
```

  - En créant une fonction proxy (ES5!)

```
var newFn = fn.bind(ctxt);
```

Les objets : this

```
global.test = 'global';

function log() {
  console.log(this.test) ;
}

var obj = {
  test : 'objet',
  log : log
}

log(); //→ global
obj.log(); // → objet

log.call(obj); // → objet
log.apply(obj); // → objet

var proxy = log.bind(obj) ;

log(); // → global
proxy(); // → objet
```

Les constructeurs

- Pour définir un objet
  - Utiliser la syntaxe littérale : { prop : value}
  - Une fonction comme constructeur avec le mot clef new
  - Dans un constructeur this représente l'objet
  - Toute fonction peut être utilisée comme constructeur

```
function contact() {
   this.nom = '';
   this.prenom = '';
   this.toString = function() {
      return this.prenom + ' ' + this.nom;
   };
}

var c = new Contact();
```

Prototype

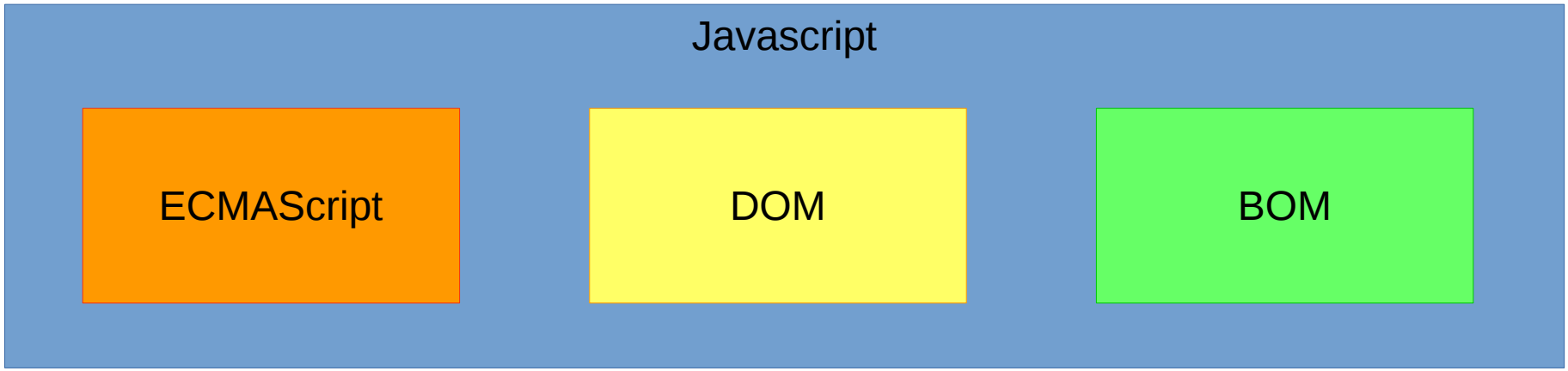
- Tout objet a forcément un prototype
  - Le prototype est lui même un objet
  - fn.prototype permet de définir le prototype des objets construits avec cette fonction comme constructeur
  - Accéder au prototype d'un objet
    - objet.__proto__ ***(deprecated)***
    - Object.getPrototypeOf(objet)
- Lorsqu'on accède à une propriété de l'objet
  - La propriété est recherchée dans l'objet
  - Puis dans son prototype et récursivement
- L'utilisation des prototypes permet de reproduire
  - Une notion de classe : le prototype contient les méthodes
  - Une notion d'héritage : avec la chaîne prototypale

Prototype

```
function Parent() {
    this.toString = function() {
    return this.prenom + ' ' + this.nom;
  }
}

function Contact() {
  this.nom = '';
  this.prenom = '' ;
}

contact.prototype = new Parent();

var contact = new Contact();
contact.nom = 'Youg';
contact.prenom = 'Angus';

console.log(contact.toString()); // → Angus Young
```

Introduction

- **JavaScript à l'origine est un langage qui s'exécute dans le navigateur**
  - ECMAScript
  - Document Object Model (DOM)



DOM

- **DOM contient des méthodes et des interfaces pour interagir avec la page Web**
- **DOM est une API basée sur un arbre pour HTML et XML**
  - SAX fourni une API orientée événement pour parser le XML
- **L'objet document est le seul qui soit commun au DOM et au BOM**
  - getElementsByTagName()
  - getElementsByName()
  - getElementById()
- **Tous les attributs HTML sont représentés par des propriétés**
  - `monImg.src = 'toto.png';`
  - `maDiv.className = 'footer'; // class → className`

XMLHTTPRequest

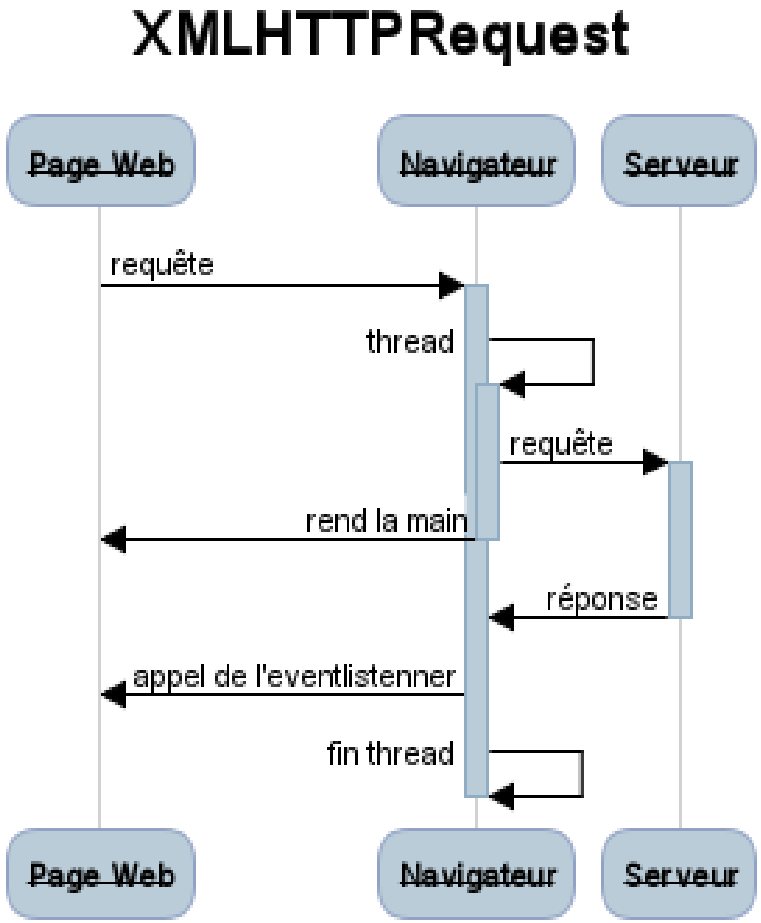
- **Permet de faire des requêtes ajax**

```
var xhr = new XMLHttpRequest();

xhr.open('POST', 'http://mon_site_web.com/ajax.php');

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send('param1=' + value1 + '&param2=' + value2);

xhr.addEventListener('readystatechange', function() {
  if (xhr.readyState === XMLHttpRequest.DONE
                  && xhr.status === 200) {
    document.getElementById('fileContent')
         .innerHTML = xhr.responseText ;
  }
});
```

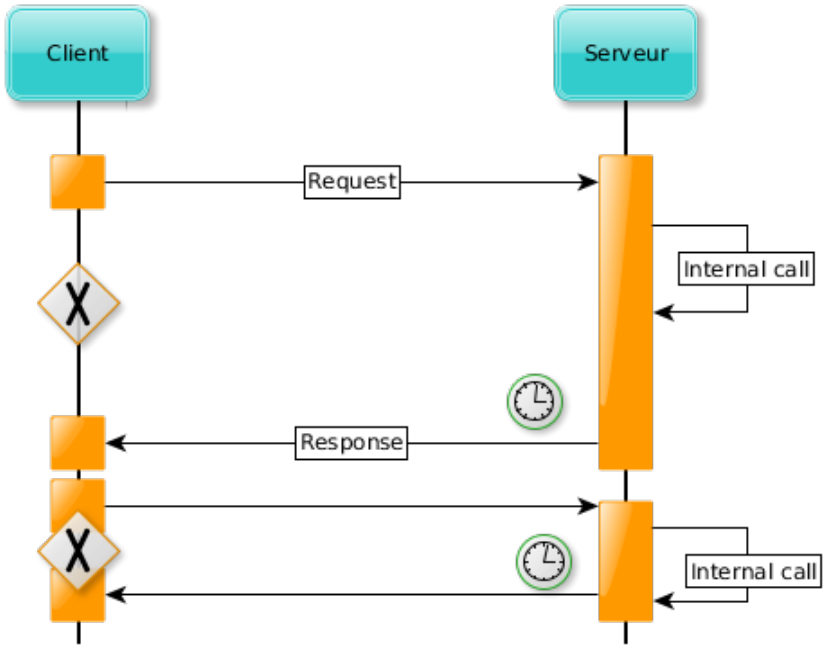
XMLHTTPRequest

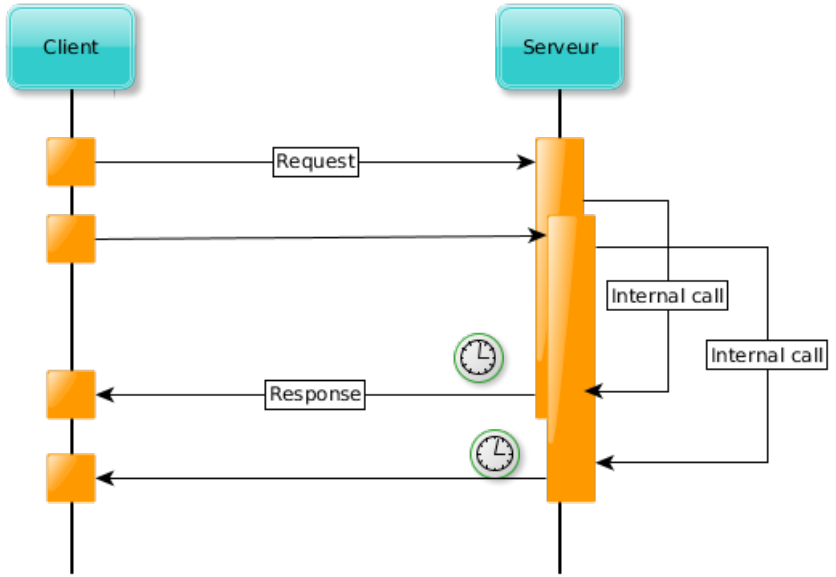
Assynchrone !!!

- **La page Web délègue la requête au navigateur**
- **Le navigateur rend la main**
- **Le navigateur appelle un bout de code de la page quand la requête est terminée**



Synchrone vs Asynchrone





Javascript

Comment l’utiliser ?

```
<!DOCTYPE html>
<html>
	<head>
	  <script>
function myFunction() {
	document.getElementById("demo")
	  .innerHTML = "Paragraph changed.";
}
	  </script>
	</head>
	<body>
	  <h1>My Web Page</h1>
	<p id="demo">A Paragraph</p>
	<button type="button"
		onclick="myFunction()">Try it</button>
	    <script src="myScript.js"></script>
	</body>
</html>
```

BOM

- **Il n’y a pas de standard pour BOM donc chaque navigateur a sa propre implémentation**


KEEP
CALM
AND
ASK
YODA
KEEP
CALM
AND
ASK
YODA