Javascript

Introduction

Créé en 1995 par Brendan Eich pour Netscape

Au depart il s'agissait d'un langage executé côté serveur : LiveScript

Adapté pour s'exécuter côté client

Langage scriptural interprété par un navigateur

But : étendre le cycle de vie d'une page web au niveau client

Repris et standardisé par l'Ecma

Pourquoi Javascript ?

Ajouter / supprimer des éléments sur la page

Modifier les css appliqués

Réagir à des évènements (clic de souris par exemple).

Valider les contenus d'un formulaire

Obtenir des informations sur le navigateur

Afficher des alertes à l'utilisateur

Etc …

Orienté objet

Pas un langage POO classique

- Basé sur des " Prototype"

Pas de réel concept de classes

- "Pseudo-classes" **collections de clés/valeurs**

Notions de bases

Où mettre mon Javascript ?

Dans la page HTML

- Head ou Body

```
<script type="text/javascript">

</script>
```

Dans un fichier avec l'extension .js

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

Ecrire dans une page

La chaine de caractère s'écrit à l'endroit de l'inclusion du script

```
<script type="text/javascript">
    document.write("Hello World");
</script>
```

Afficher une popup

```
<script type="text/javascript">
    window.alert("Hello !");

    alert("World");
</script>
```

Saisir une valeur

```
<script type="text/javascript">
    window.prompt("Choisir un chiffre", "0");

    prompt("Choisir un chiffre", "0");
</script>
```

Déclarer un tableau

Plusieurs façons de créer un tableau :

- Créer un objet **Array**
- Utiliser des crochets []

Supporte tous les types de données Javascript

```
var fruitBasket1 = new Array("Apples", "Bananas", "Pears");
var fruitBasket2 = ["Oranges", "Bananas", "Strawberries"];
var fruitBasket3 = [];

var apple = fruitBasket1[0];
fruitBasket3.push(apple);
```

Boucle *foreach*

Effectuer une opération par élément d'un tableau

- L'élément courant du tableau est représenté par la variable **element**

```
var myArray = ["Apple", "Strawberry"];
myArray.forEach(function (element) {
    console.log(element);
});
```

Fonctions

Créer une fonction

Une instruction qui contient des instructions

Déclaré avec le mot clé **function**

Peut prendre des **arguments** en paramètre

```
function maSuperFonction(param1, param2) {
    // ...
}
```

Appeler une fonction

Appel d’une fonction :

```
maSuperFonction("Hello", "World");
```

mot clé **return** afin de retourner une valeur

La valeur retournée peut ensuite être utiliser par le script

Fonction anonyme

Fonction qui n’a pas de nom

Utilisé pour des questions de « style »

Allège le code

Fonction qui ne compte pas être réutilisée

Fonctions pratiques

setTimeout(function, delay [, params…]);

La méthode **setTimeout** définit une action à exécuter et un délai avant son exécution. Elle retourne un identifieur pour le processus

```
var sto = setTimeout(function () {
    console.log("Hello world");
}, 5000);
```

Il est possible d'interrompre le décompte

```
clearTimeout(sto);
```

Fonctions pratiques

setInterval(function, delay [, params…]);

Similaire à *setTimeout*, elle déclenche répétitivement la même action à intervalles réguliers.

```
var sin = setInterval(function () {
    console.log("Hello world");
}, 2000);
```

Il est possible d'interrompre l'interval

```
clearInterval(sin);
```

Intéragir avec le DOM

Présentation

Document Object Model

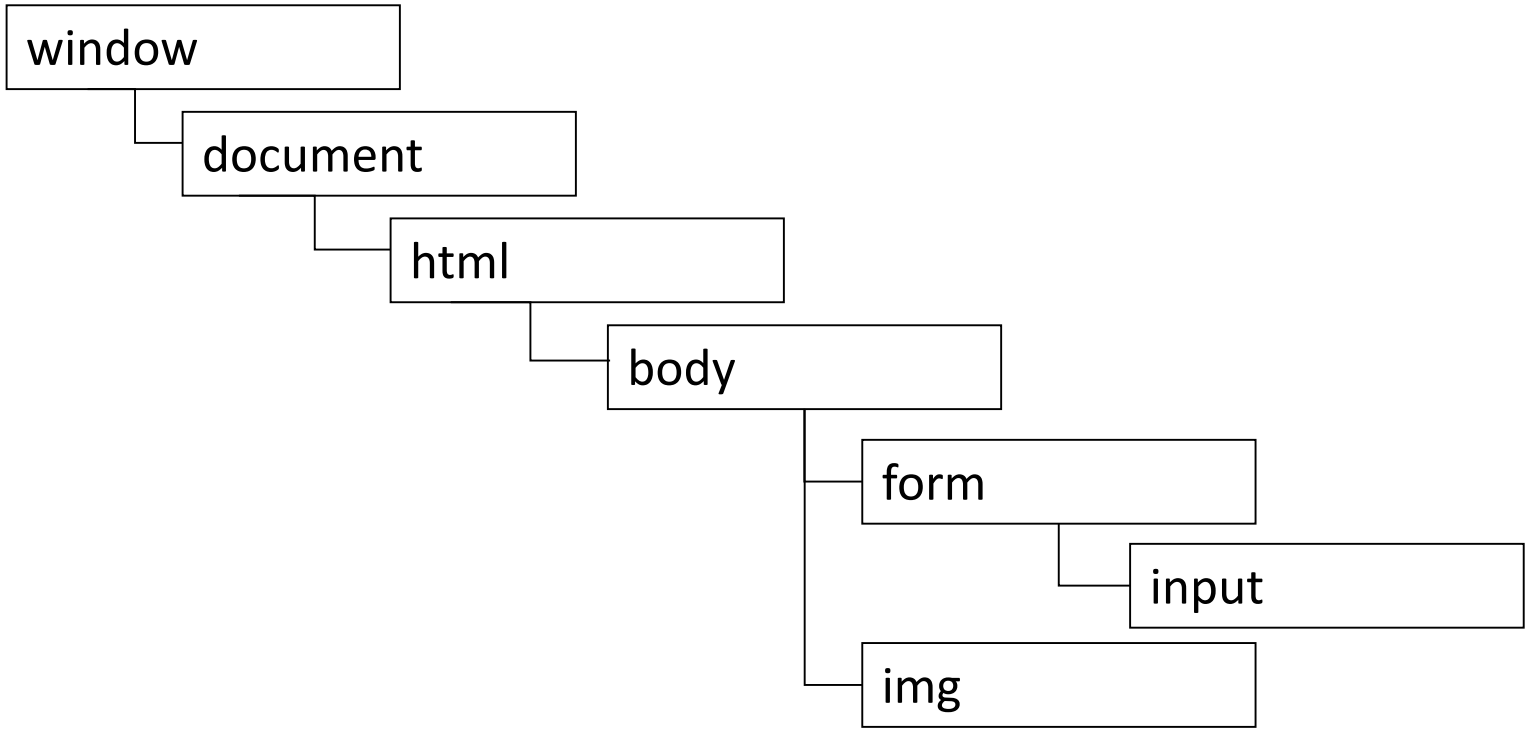
W3C Standard

- Outils standardisés pour accéder et manipuler le document html

Indépendant du langage ou de la plateforme

Chaque noeud représente un objet

DOM Arborescence



Accéder aux éléments

Accès à toute la structure de la page HTML

Permet de dynamiquement :

- Accéder aux éléments HTML
- Modifier / supprimer / créer des attributs et valeurs
- Organiser les éléments hiérarchiquement

Accéder aux élements

Accès via l'ID:

- Retourne un unique élément

```
document.getElementById(id);
```

Accès via la classe

- Retourne un tableau d'éléments

```
document.getElementsByClassName(class);
```

Accéder aux éléments

Accès aux éléments grâce à leur tags:

- Retourne un tableau d'éléments

```
document.getElementsByTagName(tagName);
```

Accès aux éléments par leur « name »

- Retour les homonymes sous forme de tableau

```
document.getElementsByName(name);
```

Accéder aux éléments

Accès à tous les noeuds fils :

```
element.childNodes;
```

Accès au noeud parent :

```
element.parentNode;
```

Manipuler les attributs

Accès aux attributs d'un élément :

```
element.getAttribute("attribut");
```

Modifier l'attribut d'un élément :

```
element.setAttribute("attribut", "valeur");
```

Manipuler les valeurs

Accès au texte d'un élément :

```
element.textContent;
```

Modifer le texte d'un élément :

```
element.textContent = "texte";
```

Créer un élément

Créer un élément

```
var e = document.createElement('p');
```

Modifier un élément

```
e.style.textAlign = 'center';
```

Ajouter un élément dans le DOM

Ajout de l'élément à un parent

```
parent.appendChild(e);
```

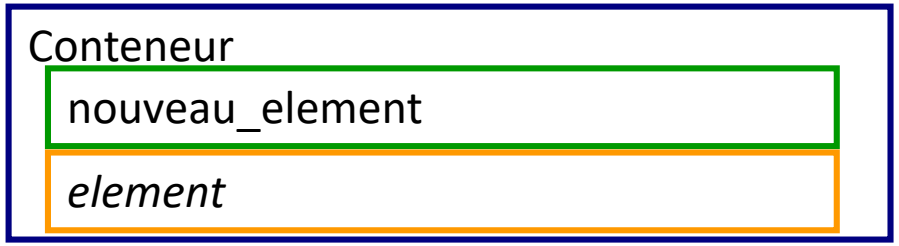
Supprimer un élément

```
var e = document.getElementById("deleteMe");
        e.parentNode.removeChild(e);
```

Ajouter un élément dans le DOM

Ajouter un élément avant un autre

```
element.parentNode.insertBefore(nouveau_element,element);
```



Api Selector 2

Une nouvelle API existe afin de faciliter nativement les recherches dans le DOM :

- var alerts = document.querySelectorAll("p.warning, p.error");
- var x = document.querySelector("#bar, #foo");

Evènements

Présentation

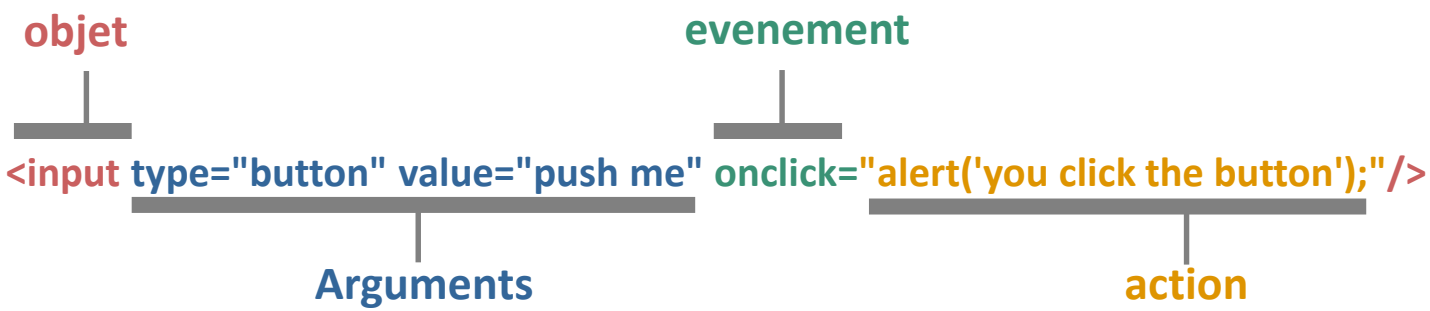
Quand :

- Intéraction utilisateur
- Action dans le contexte d’exécution

Dépend d’un “objet”

Peux appeler des fonctions

Affectation HTML



Liste des évènements

Evenement	Description
onLoad	La page est complétement chargée
onUnload	Le navigateur quitte la page courante
onClick	Clic sur un élément
onDblclick	Double clic sur un élément
onMouseover	Souris qui passe au dessus d’un élément
onMouseout	Souris quitte un élément

Liste des évènements

Evenement	Description
onFocus	Un champ recoit le focus
onBlur	Un champ perd le focus
onChange	La valeur d'un champ change
onSelect	Le texte d'un champ est sélectionné
onSubmit	Quand un formulaire est soumis

Gérer les évènements

Il y a différentes possibilités afin de gérer les événements :

```
<button id="test" onclick="alert('Click !')">Afficher un message</button>
```

devient :

```
document.getElementById('test').onclick = function () {
    alert('Click !');
};
```

Gérer les évènements

Ou alors, il est possible d'ajouter une fonction « d'écoute » à un objet.

- Un objet peut avoir plusieurs fonctions « d'écoutes »

- **addEventListener(eventName, listenerFunction, bubbles)**

```
document.getElementById('test').addEventListener("click", function () {
    alert('Click !');
}, false);
```

- **removeEventListener(event, listenerFunction, bubbles)**

Orienté objet

Présentation

JavaScript est un langage orienté objet utilisant les Prototypes

Objects en JavaScript sont des collections de clés / valeurs

number, *string*, *boolean*, *null* et *undefined* sont les types primitifs

Les Arrays sont des objects

Présentation

Un objet contient des propriétés

Une propriété a un nom et une valeur

- Nom : string
- Valeur : n'importe quelle valeur javascript
  - Strings
  - Arrays
  - Functions!

Il y a plusieurs méthodes pour créer un objet

- **Object Literals (objets literaux)**

Objet littéral

Notation simple pour créer un objet

```
var barney = {
    "firstName": "Barney",
    "lastName": "Stinson",
    "saySmthg": function() {
        console.log("It's gonna be...");
    }
}
```

Objet littéral

Les guillemets autour de la prioriété sont optionels si le nom n’est pas un mot clé javascript

Les valeurs peuvent être également des objets

Exemple

```
var trip = {
    departure: {
        city: "Paris",
        country: "France"
    },
    arrival: {
        city: "Montreal",
        country: "Canada"
    },
    price: 890
}
```

Objet littéral

Pour accéder à une propriété :

```
var firstName = barney.firstName;

var lastName = barney["lastName"];
```

Pour appeler une méthode:

```
barney.saySmthg();

barney.["saySmthg"]();
```

Objet littéral

Clone d'un objet existant:

- `Object.create`

```
var anotherBarney = Object.create(barney);

anotherBarney.saySmthg = function() {
    console.log("... Legendary!");
};

barney.saySmthg();
anotherBarney.saySmthg();
```

Parcourir les clefs

Il est possible de parcourir les clefs d'un objet littéral

```
for(var prop in barney) {
    console.log(prop);
    console.log(barney[prop]);
}
```

Le mot clef *new*

Vous pouvez également utiliser le mot clé new dans certains cas

```
var anArray = new Array(); // Ca marche

var newBarney = new barney();  // Ca ne marche pas, car Barney est un
                               // objet litteral
```

Fonctions avancées

Expression de fonction

JavaScript supporte également les expressions de fonctions

- Fonctions avec ou sans nom (anonyme)

```
var calculCarre = function (x) {
    console.log(x * x);
}

console.log(calculCarre(3));
```

Closure

Permet à une fonction d'avoir des variables privées

Permet de renvoyer une fonction

La fonction imbriquée possède toujours l'accès au « scope » du parent

```
function changeColor(color) {
    return function () {
        document.body.style.color = color;
    }
}
```

IIFE

Immediatly **I**nvoked **F**unction **E**xpression

Reprend le principe des fonctions d'expressions et des closures

Permet de créer des scripts avec des variables privées

```
var test = (function () {
    var a = 1;
    return {
        getA: function () {
            return a;
        }
    }
})();

console.log(typeof a); // Undefined
console.log(test.getA()); // 1
```

Déclaration VS Expression

La déclaration des fonctions sont évaluées avant les autres instructions

Les fonctions d’expressions sont évaluées après les instructions précédentes

First-class functions

JavaScript est aussi un language fonctionnel

First-class functions:

- Être assignées à des variables
- Être passé comme paramètre à d'autres fonctions
- Peux être retourné depuis une fonction