Formation Développeur Expert Java

Introduction à JavaScript

- Création d'IOcean en 2001
- 2 Implantations :
  - Siège à Montpellier
  - Agence Paris
- 2 Activités :
  - ESN (SSII)
  - Editeur
- En 2016, 30 personnes pour 2,2 M€

Votre formateur

IOCEAN

Les projets informatiques dont les clients parlent avec plaisir

Sommaire

- Les grandes lignes
- Déclaration et utilisation côté client
- Les types
- Les variables
- Les opérateurs
- Les structures de contrôle
- Les types en détail
- Les fonctions
- Tout est objet
- Le DOM et ses interactions

Les grandes lignes

Quand ?

Pourquoi ?

Script ?

Compilé ou interprété ?

Typé ?

Objet ?

Client ?
Serveur ?

Lien avec Java ?

Standardisé ?

Les grandes lignes

Quand ?	1995, chez Netscape, par Brendan Erich
Pourquoi ?	Dynamiser les sites web, permettre plus d'interactivité
Script ?	Langage léger, s'exécutant dans un environnement hôte
Compilé ou interprété ?	Interprété, c'est un langage dynamique
Typé ?	Oui : classé dans la catégorie du typage léger
Objet ?	TOUT EST OBJET (excepté les types de base)
Client ? Serveur ?	Initialement client, aujourd'hui client et serveur
Lien avec Java ?	Pas grand-chose à part le nom et quelques éléments de syntaxe…
Standardisé ?	ECMAScript

Un exemple côté client

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Médiathèque</title>
</head>
<body>

<h1>Bienvenue !</h1>
<span></span>

<script>
    function hello(prenom) {
        let message = "Hello " + prenom + " !";
        console.log(message);

        let span = document.querySelector("span");
        span.textContent = message;
    }

    hello("Marc");
</script>

</body>
</html>
```

Déclaration et utilisation côté client

Déclaration avec la balise <script> (dans <head> ou <body>)

• script local à la page HTML

```
<script>
    alert("Code dans la page");
</script>
```

• script par fichier externe

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

Bonnes pratiques

- Toujours préférer un fichier externe (séparation des responsabilités)
- Toujours préférer le chargement JS en bas de page

Déclaration via les gestionnaires d'événèments des balises HTML

```
<button onclick="alert('Hello world!')">
```

Les types

Javascript définit 7 types, dont 6 primitifs

Les **nombres**	19 18.01
Les **chaines de caractères**	"une chaine"
Les **booléens**	true false
Les **symboles**	Symbol(123456) Unique et non modifiable, utilisé par exemple pour représenter des identifiants
null	Utilisé pour indiquer une variable explicitement nulle
undefined	Utilisé pour indiquer une variable non définie, non assignée
Les **objets**	TOUT est objet Number, String, Boolean, Function, Array, Error, Date, …

Les variables

Une variable est un nom symbolique représentant une valeur (typée) utilisée dans le code.
Déclaration de variable :

- **var** unNombre = 42;
- **let** vrai = true;
- **const** JANVIER = "Janvier";

Un nom (ou identifiant) de variable doit commencer par une **lettre**, un _ ou un **$**
Attention JS est sensible à la casse : azerty != AzerTY

Pourquoi plusieurs déclarations possibles ?
Une histoire de portée et d'évolution du langage

- **var** : déclaration historique, portée globale ou locale à une fonction
- **let** : permet de placer la variable dans une portée de bloc {…}
- **const** : permet de placer la variable dans une portée de bloc {…} et d'interdire le réassignement

Les opérateurs

+ - / * % …

+= -= *= /= ++ -- …

> < >= <= == != === !== …

- Les opérateurs d'affectation
- Les opérateurs de comparaison
- Les opérateurs arithmétiques
- Les opérateurs binaires
- Les opérateurs logiques
- Les opérateurs de chaînes
- L'opérateur conditionnel
- L'opérateur virgule
- Les opérateurs unaires
- Les opérateurs relationnels

Comment vérifier l'égalité ?
== et **!=**
Egalité ou inégalité **simple** avec conversion de type automatique

=== et **!==**
Egalité ou inégalité **stricte** sans conversion de type

Les structures de contrôle

```
if (condition) {
  … code…;
} else {
 … code…;
}
```

Condition "falsy" :

- false
- undefined
- null
- 0
- NaN
- la chaîne de caractères vide ("")

```
switch (expression) {
  case valeur1:
    …code…
    [break;]
  case valeur2:
    …code…
    [break;]
  default:
    …code…
    [break;]
}
```

Les structures de contrôle

Les boucles

- **for**
- **while**
- **do**...**while** : au moins une instruction
- **for**...**in** : itérer sur les propriétés
- **for**...**of** : itérer sur les valeurs

- tableau.**forEach**()

Les structures de contrôle

```
try {
   throw "uneException";
} catch (e) {
   alert(e);
} finally {
   alert("fin dans tous les cas");
}
```

Les nombres

Les nombres sont implémentés selon des nombres à virgule flottante…
Il n'y a donc pas de type spécifique pour un nombre ENTIER en Javascript.

```
let nombre = 13;
let autreNombre = 11.23;
let encore = new Number(19);

let resultat = 0.1 + 0.2;
// resultat = ???
```

JS permet de travailler avec 4 littéraux numériques :

- décimal (13, 13.002)
- binaire (0b10 ou 2 en base 10)
- octal (0o10 ou 8 en base 10)
- hexadécimal (0xF ou 15 en base 10)

L'objet **Number** permet de manipuler les nombres comme des objets

L'objet **Math** natif permet d'appliquer de très nombreuses fonctions sur les nombres

Les chaines de caractères

Une chaine de caractère JS est codé en UTF-16.
Une chaine est également **immuable**.

```
let chaine = "une chaine";
let chaine2 = 'chaine2';
let autreChaine = new String("une chaine");

let tailleChaine = chaine.length;

let caractere = chaine[0];
let autreCaractere = chaine.charAt(0);
```

De nombreuses méthodes de l'objet **String** existent pour manipuler les chaines 

Conversion de types

JS est un **langage dynamique** où il n'est pas nécessaire de préciser le type de la variable.
Le type de la variable sera défini automatiquement à l'exécution du script.

```
let nombre = 12;                        // typeof nombre = "number"
nombre = "une chaine désormais";        // typeof nombre = "string"
```

Attention aux conversions automatiques

```
let a = "12"  + 12;
a = ?

let b = 12 + 12 + "12";
b = ?
```

Les tableaux

Un tableau JS est une simple liste indexée, c’est-à-dire qu'on accède à l'un de ses éléments via son index.

```
let tableau = [1,2,3,4,5,6]; // préférer la notation littérale
let tableau2 = new Array(1,2,3,4,5,6);
let tableau3 = Array(1, "aaa", 3, 4.82, "b");

let premier = tableau[0];
let dernier = tableau[tableau.length - 1];
```

L'objet **Array** propose de nombreuses fonctions pour manipuler les tableaux 

Les collections avec clés

Collection clé-valeur

```
let uneMap = new Map();
uneMap.set("Marc", 10);
uneMap.set("Delphine", 13)

uneMap.has("Chloé");
uneMap.get("Marc");
uneMap.size;
uneMap.delete("Delphine");
```

Les ensembles (clé unique)

```
let unSet = new Set();
unSet.add("Marc");
unSet.add("Delphine");
unSet.add("Marc");

unSet.size; // = 2
```

Les objets **Map** et **Set** permettent de manipuler des collections à clé.
Leurs clés peuvent être de tous types et leur parcours se fait selon leur ordre d'insertion.

Les fonctions

Déclaration de fonction

```
function maFonction(arg, arg) {
    …code…
}
```

Expression de fonction

```
var uneFonction = function maFonction() {
};
```

Expression de fonction anonyme

```
var uneFonction = function() {
}
```

Fonction imbriquée

```
function maFonction() {
    …code…
    function imbriquee() {
        …code…
    }
}
```

Les fonctions

Lors de l'appel d'une fonction, les arguments de la fonction sont passés **par valeur.**

Une fonction a toujours une valeur de retour :

- soit explicite via l'instruction **return**,
- soit **undefined**

La **portée** d'une fonction est la fonction dans laquelle elle est déclarée ou le script entier.

Une fonction a toujours accès aux fonctions et variables qui appartiennent à la portée dans laquelle elle est définie.

Une fonction imbriquée a toujours accès au contexte de sa fonction parente

Tout est objet

Qu'est-ce qu'un objet ?

Un **objet** est un ensemble de **propriétés** : une clé associée à une valeur.

Le nom de la propriété (clé) est une chaine de caractères (ou doit être convertible en chaine).

La valeur peut être de tous types.

```
// Création d'un objet, notation littérale
var monObjet = {
    taille: 10,
    couleur: "jaune",
    forme: "rectangle"
};
```

```
// Création via constructeur générique
var monObjet = new Object();
monObjet.taille = 10;
monObjet.couleur = "jaune";
monObjet.forme = "rectangle";
```

```
// Création via constructeur spécifique
function Element(taille, couleur, forme) {
    this.taille = taille;
    this.couleur = couleur;
    this.forme = forme;
}

var monObjet = new Element(10, "jaune",
"rectangle");
```

Tout est objet

On accède aux propriétés de l'objet par le séparateur **.** ou via des crochets à la manière d'un tableau :
• monObjet.maPropriete
• monObjet["maPropriete"]

On appelle les fonctions de l'objet de la même façon, dénommées **méthode** pour l'occasion **:**
• monObjet.maFonction()
• monObjet.["maFonction"]()

```
var monObjet = {
    taille: 10,
    couleur: "jaune",
    forme: "rectangle",
    afficherProps: function() {
        console.log(this.taille, this.couleur, this.forme);
    }
};
```

Tout est objet

this : dépendant du contexte

• Dans le contexte global, **this** fait référence à l'objet global (dans le navigateur, this === window)

• Dans le contexte d'une fonction, **this** fait référence à **l'objet appelant (**sur lequel on appelle la fonction)

➢ si l'objet appelant n'est pas spécifié, **this** sera l'objet global en mode non strict, ou undefined en mode strict

Accesseurs

Créer des propriétés qui retournent des valeurs dynamiques…

```
var gestionnaireDate = {
    get maintenant() {
        return new Date()
    }
}
gestionnaireDate.maintenant
```

Mutateurs

Exécuter une fonction à chaque modification de la valeur d'une propriété

```
var temps = {
    secondes: 0,
    set minutes(arg) {
        this.secondes = arg * 60;
    }
}
temps.minutes = 123;
(temps.secondes = 7380)
```

Le modèle objet

JS est un langage objet basé sur les **prototypes** (et non les classes).

Un prototype, qui est également un objet, permet de partager des propriétés avec tous les objets qui l'utiliseront.

A l'exécution, JS cherchera la propriété de l'objet courant et remontera la chaine de prototype jusqu'à la trouver ou non

```
function Plan(longueur, largeur) {
    this.longueur = longueur;
    this.largeur = largeur;
}
var plan = new Plan(20, 10);
Plan.prototype.afficher = function() {
    console.log("Plan d'une longueur de", this.longueur , "et d'une largeur de", this.largeur);
}
plan.afficher(); // Plan d'une longueur de 20 et d'une largeur de 10
```

Bon à savoir

La remontée de variable et de fonction (hoisting)

```
console.log(maVariable);    // Erreur ou pas ?
var maVariable = 10;
console.log(maVariable);
```

Les variables globales : correspondent à des propriétés de l'objet global

Dans un script, hors d'une fonction : var a = 10; est équivalent à window.a = 10;

Travailler avec un nombre indéfini d'arguments :

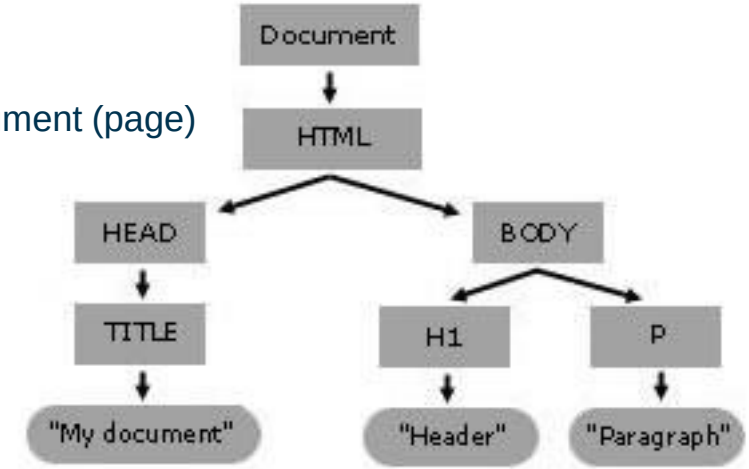
- L'objet **arguments** : objet spécifique permettant d'accéder à tous les arguments d'une fonction
- Le **paramètre du reste** : tableau contenant tous les arguments non nommés

```
function (arg1, arg2, ...args) {…}
```

Le DOM

DOM : Document Object Model

En JS le DOM est représenté par une structure objet arborescente, accessible par la variable **document** (de l'objet global window)



Il permet d'interagir **dynamiquement** sur l'ensemble du document (page)

- Créer de nouveaux éléments
- Modifier des éléments
- Supprimer des éléments
- Modifier l'apparence et l'organisation

En bref, tout est modifiable…

https://developer.mozilla.org/fr/docs/Utilisation_du_DOM_Level_1_Core_du_W3C

Structure du DOM

L'objet **window** est l'objet global (dans un navigateur)

Il permet d'accéder, entre autre, à toutes les propriétés de la fenêtre du navigateur et toutes les propriétés globales.

L'objet **document** est directement accessible et est l'équivalent de **window.document**

Tous les nœuds d'un document sont représentés par des objets implémentant des interfaces :

- EventTarget
- Node
- HTMLDocument
- HTMLElement
- HTMLFormElement
- HTMLInputElement
- HTMLTableElement
- HTMLImageElement

Sélectionner des éléments

Approche moderne et recommandée, basée sur les sélecteurs CSS

document.**querySelector**(".classe") : retourne le premier élément trouvé

document.**querySelectorAll**("div.info") : retourne un table NodeList de tous les éléments trouvés

S'applique de la même façon sur un élément pour trouver des éléments descendants

element.**querySelectorAll**("p")

Ancienne approche

document.**getElementById**("id")

document.**getElementByTagName**("div")

document.**getElementsByClassName**("uneClasse")

Créer et supprimer des éléments

document.**createElement**("div")
document.**createTextNode**("du texte tout simplement")

parentElement.**appendChild**(autreElement)
parentElement.**removeChild**(autreElement)
parentElement.**replaceChild**(nouvelElement, ancienElement)
parentElement.**insertBefore**(nouvelElement, referenceElement)

…

Modifier le DOM

monElement.**setAttribute**("class", "maClasse")
monElement.**style.color** = "red";
document.body.**style.backgroundColor** = "yellow";

element.**innerHTML** permet de lire le contenu HTML d'un élément et de le remplacer
element.**innerHTML** = " Mon nouveau contenu ";

element**.textContent** permet de lire le contenu textuel d'un élément et de le remplacer
element.**textContent** = "Du texte et seulement du texte";

…

La gestion des évènements dans le DOM

```
<div>
   <button onclick="console.log('1')">Premier</button>
   <button id="btn2">Deuxième</button>
   <button id="btn3">Troisieme</button>
</div>
```

```
var btn = document.querySelector("#btn2");
btn.addEventListener('click', function() {
    console.log("Via addEventListener");
});
```

3 possibilités pour gérer les évènements…

Laquelle préférer ?

```
btn = document.querySelector("#btn3");
btn.onclick = function(event) {
    console.log("Via propriété onClick");
};
```

Liens utiles

https://developer.mozilla.org/fr/

https://www.w3schools.com/js/default.asp

https://www.ecma-international.org/ecma-262/8.0/index.html

https://mbeaudru.github.io/modern-js-cheatsheet/

http://overapi.com/javascript

Restons en contact.

DIGINAMIC
Lionel Cabon, Directeur
contact@diginamic.fr

N° Déclaration OF : 91 34 08867 34
Nantes, Paris, Montpellier
www.diginamic.fr