JavaScript

JavaScript

C’est quoi ?

JavaScript, c’est quoi ?

- JavaScript est un langage de programmation de script orienté objet.
- Code source Interprété côté client.
- Créé en 1995 par Brendan Eich.
- ECMAScript (ES) est l’ensemble des normes d’interprétation du JS.
- Dernière version de JS est **JavaScript ECMAScript 6**

JavaScript

Quand l’utiliser

JavaScript, quand l’utiliser

JavaScript est utilisé pour exécuter des scripts sur le terminal client.

- Ajouter de l’interaction avec le document.
- Ajouter des fonctionnalités au navigateur
- Manipuler des éléments HTML.
- Manipuler des chaîne.
- ajouter des animations...

JavaScript

Intégration

Les scripts interne

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

        // on écrit ici le code JS

</script>
```

- La balise <script /> peut être placée n'importe où dans le document HTML.
- Le contenu des balise <script /> doit impérativement être JS.
- L'attribut `language` permettra de préciser la version de JS utilisé.

Les scripts externe

- Les fichiers portent l'extension .js

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

- Si on utilise l'attribut `src` de la balise `script`, les balises script ne doivent pas contenir de code

Ou placer les scripts

- On place les balise de script ou on veut dans le document.
- Dans les bonnes pratiques et pour des raison d’optimisation du chargement des pages, on placera les scripts en fin de document juste avant la balise </body>

JavaScript

Les instructions

Syntaxe

- Les instructions sont séparées par des point-virgules.
- Elles peuvent être écrite sur une seule ligne.
- Les espaces ne gênent pas leurs exécution.

```
instruction_a;

instruction_b; instruction_c;                instruction_d;
```

label

- Nouveauté ES6
- Utilisé avec les instruction `break` ou `continue`.
- Exemple

```
boucle1:
for ( … ) {
    if ( … )  {
            break boucle1;
    }
}
```

JavaScript

Les commentaires

- Les commentaires donnent des informations aux développeurs sur le code écrit.
- Ils ne sont pas interprété par les moteurs de rendu.

```
// Un commentaire sur une seule ligne

/* Un commentaire
sur plusieurs lignes */
```

JavaScript

Les variables

Les variables : Syntaxe

- Déclaration simple : `var maVariable;`
- Déclaration avec affectation : `var maVariable = “Hello”;`
- Déclaration de plusieurs variables : `var varA, varB = 42, varC = “World”;`

Les variables : Portée

- Une variable aura une **portée globale lorsqu'elle est déclarée hors d'une fonction**.
- Une variable aura une **portée locale lorsqu'elle est déclarée dans une fonction**.

Les variables : Durée de vie

- Une variable locale (à une fonction), vie le temps d'exécution de la fonction.
- Une variable globale (à la page), vie le temps d’exécution de la page (tant que la page est affichée).

JavaScript

Les opérateurs

Les opérateurs Arithmétique

- `+` Addition
- `-` Soustraction
- `*` Multiplication
- `/` Division
- `%` Modulo (retourne le reste d'une division)
- `++` Incrémentation
- `--` Décrémentation

Les opérateurs d’Assignation

L’opérateur d’assignation (=) assigne une valeur à une variable.

- `=`
- `+=` `x += y;` équivaut à `x = x + y;`
- `-=` `x -= y;` équivaut à `x = x - y;`
- `*=` `x *= y;` équivaut à `x = x * y;`
- `/=` `x /= y;` équivaut à `x = x / y;`
- `%=` `x %= y;` équivaut à `x = x % y;`

Les opérateurs de chaîne

- Seul le + peut être utilisé comme opérateur de chaîne.

```
var str1 = “Hello”;

var str2 = “World”;

var str3 = str1 + “ “ + str2; // Hello World

var str4 = “Goodbye”;

    str4 += “ “+str2; // Goodbye World
```

Ajouter des chaînes et des nombres

- `var x = 21 + 21; // 42`
- `var y = “21” + 21; // 2121`
- `var z = “Trillian” + 21; // Trillian21`

Les opérateurs de comparaison

- == égale à *(valeur)*
- === strictement égale à *(valeur ET type)*
- != différent *(valeur)*
- !== strictement différent *(valeur OU type)*
- > supérieur à
- < inférieur à
- >= supérieur ou égale à
- <= inférieur à égale à
- ? opérateur ternaire

Les opérateurs logiques

- `&&` ET logique
- `||` OU logique
- `!` NON logique

JavaScript

Les types de données

Les chaînes

- avec double quote : `var string1 = “Hello world”;`
- avec simple quote : `var string2 = ‘Goodbye world’;`

Les nombres

- Ceci est un nombre : `var number1 = 42;`
- Ceci n’est pas un nombre : `var number2 = “42”;`

Les booléen

- Les booléens ne peuvent avoir que 2 valeurs:
  - `var x = true;`
  - `var y = false;`

Les tableaux

- Voici un tableau : `var x = new Array();`
- Voici un second tableau : `var y = [];`
- Voici un tableau avec des entrées :

```
var z = [“HTML”, “JavaScript”, “PHP”, 42];
```

- Les éléments du tableau sont séparés par des virgules.
- L’index de base des tableaux est [0].
- Accéder à une donnée : `z[1]; // JavaScript`

Les objets

- Voici un objet : `var x = new Object();`
- Voici un second object : `var y = {};`
- Voici un objet avec des données : `var z = { firstname:”Homer”, lastname:”Simpson”};`
- Accéder à une donnée : `z.firstname; // Homer`

Les valeurs indéfinis

- Une variable avec une valeur non définie :
  - var x;
  - var x = undefined;

Les valeurs null

- Une valeur nulle : `var x = null;`
- Le type de `null` est un objet.

La différence entre `null` et `undefined`

- null et undefined valent la même chose
  - `null == undefined; // true`
- mais null et undefined ne valent strictement pas la même chose
  - `null === undefined; // false`

Les valeurs vide

- Une valeur vide : `var x = “”;`
  - n’est pas nulle.
  - n’est pas indéfinie.
  - c’est une chaîne vide.

JavaScript

Les conditions

Les conditions

- Une condition est une instruction
- Elle permet de tester si une condition est réalisée ou non
- Le test de la condition retourne `true` ou `false`.

Les conditions : if (Syntaxe)

```
if ( condition à analyser ) {

        // Code exécuté si la condition est vraie.

}
```

Les conditions : if … else (Syntaxe)

```
if ( condition à analyser ) {

        // Code exécuté si la condition est vraie.

}

else {

        // Code exécuté si la condition est fausse.

}
```

Les conditions : if … elseif … else (Syntaxe)

```
if ( première condition à analyser ) {
        // Code exécuté si la première condition est vraie.
}
elseif ( seconde condition à analyser ) {
        // Code exécuté si la seconde condition est vraie.
}
else {
        // Code exécuté si aucune des conditions n'est vraie.
}
```

JavaScript

Le commutateur

L’instruction : switch

- Le commutateur `switch` et équivalent à la condition `if … elseif … else`.
- il est :
  - plus lisible
  - plus rapide à écrire
  - plus rapide à exécuter

L’instruction : switch (syntaxe)

```
switch ( condition à tester ) {
       case Valeur1:
       /* code à exécuter la condition == Valeur1 */
       break;
       case Valeur2:
       /* code à exécuter la condition == Valeur2 */
       break;
       default:
       /* code à exécuter la condition != Valeur1 &&
    condition != Valeur2 */
    break;
}
```

JavaScript

Les incrémentations

Les incrémentations

- Incrémenté, c’est ajouter 1 à la valeur précédente :

```
var x = 0; // x vaut 0;
    x = x + 1; // x vaut 1
    x = x + 1; // x vaut 2

var y = 0; // y vaut 0;
y++; // y vaut 1;
y++; // y vaut 2;
```

Les décrémentations

- décrémenté, c'est soustrait 1 à la valeur précédente :

```
var x = 0; // x vaut 0;
    x = x - 1; // x vaut -1
    x = x - 1; // x vaut -2

var y = 0; // y vaut 0;
y--; // y vaut -1;
y--; // y vaut -2;
```

Post incrémentation/décrémentation

- post incrémentation : on affecter la valeur avant d'ajouter 1.
- post décrémentation : on affecter la valeur avant de soustraire 1.

```
var x = 1 ;

    x; // x vaut 1

    x++; // vaut 1

    x++; // vaut 2

    x; // x vaut 3
```

Pré incrémentation/décrémentation

- pré incrémentation : on ajoute 1 avant d'affecter la valeur.
- pré décrémentation : on soustrait 1 avant d'affecter la valeur.

```
var x = 1 ;

    x; // x vaut 1

    ++x; // vaut 2

    ++x; // vaut 3

    x; // x vaut 4
```

JavaScript

Les boucles

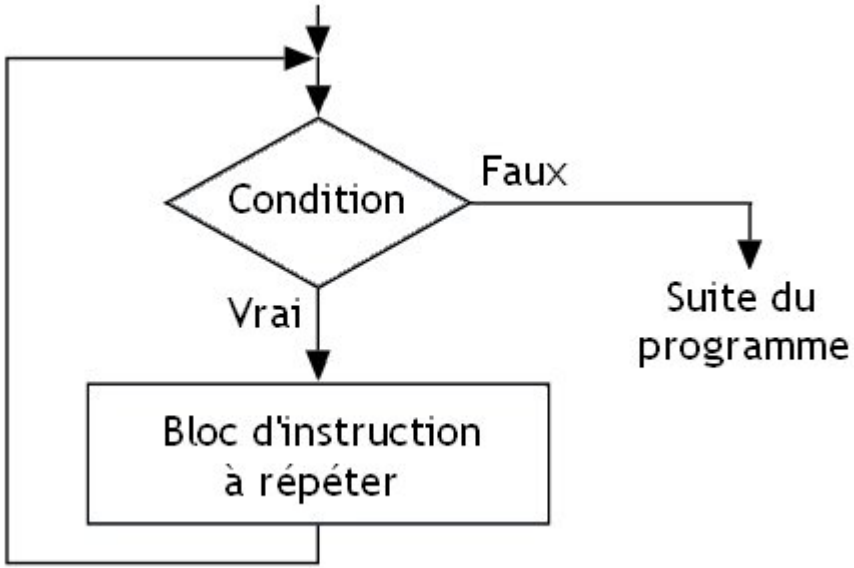
Les boucles

- Un boucle répète une action tant que la condition est satisfaite.
- Dans quels cas utiliser des boucles :
  - Pour parcourir des tableaux.
  - Pour répéter des tâches.

```
alert(1);
alert(2);
alert(3);
alert(4);
alert(5);
   ...
```

Les boucles : While

- Exécute la boucle tant que la condition est vérifiée.

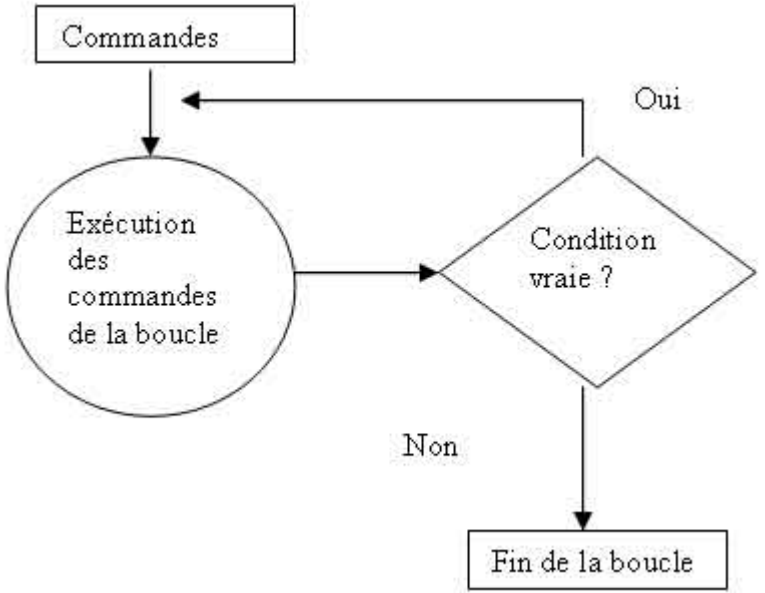


Les boucles : While (Syntaxe)

```
while( condition de bouclage )
{
   // instructions
}
```

Les boucles : Do … While

- Do … While va d'abord exécuter un bloc d'instruction, avant de tester la première condition de bouclage.

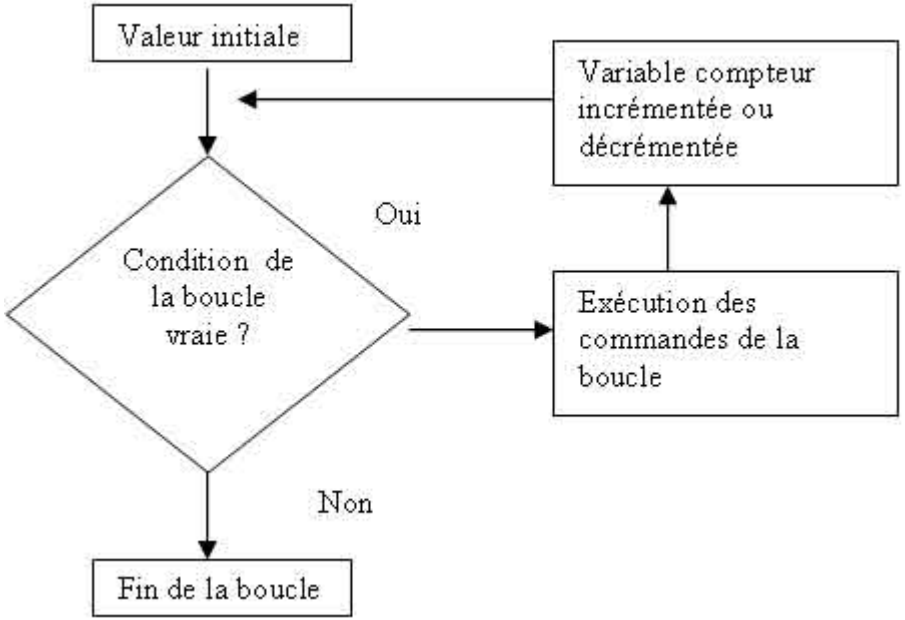


Les boucles : Do … While (Syntaxe)

```
do {
      // instructions;
} while( condition de bouclage )
```

Les boucles : For

- Boucle avec incrémentation d’une valeur de bouclage.
- à besoin de 3 paramètres :
  - initialisation
  - condition
  - incrémentation



Les boucles : For (Syntaxe)

```
for( initialisation ; condition ; incrementation )
{
    // instructions;
}
```

Les boucles : For … in

- Parcourir un objet

```
for ( entrée in objet ) {

        // instruction;

}
```

Les boucles : Break

- `break` arrête l’exécution de la boucle et continu le programme.

JavaScript

Sortie

Les données en sortie avec JavaScript

Plusieurs possibilité pour JavaScript peut “afficher” des données.

- ➔ Ecrire dans une boîte de dialogue avec `window.alert()`
- ➔ Ecrire dans la sortie HTML avec `document.write()`
- ➔ Ecrire dans un élément HTML avec `innerHTML`
- ➔ Ecrire dans la console du navigateur avec `console.log()`

window.alert()

Créer une boite de dialogue de type alert contenant le message de sortie.

```
<!DOCTYPE html>
<html>
<body>

     <h1>My First Web Page</h1>
     <p>My first paragraph.</p>

     <script>
          window.alert(5 + 6);
     </script>

</body>
</html>
```

document.write()

Tant que le document n’est pas complètement chargé, document.write va écrire la sortie à la suite.

```
<!DOCTYPE html>
<html>
<body>
     <h1>Titre de la page</h1>
     <p>Ceci est un paragraphe HTML</p>
     <script>
          document.write(5 + 6);
     </script>
</body>
</html>
```

document.write()

Si document.write est appelé après la fin de chargement du document, il écrasera la totalité du document.

```
<!DOCTYPE html>
<html>
<body>
     <h1>Titre de la page</h1>
     <p>Ceci est un paragraphe HTML</p>

     <button onclick="document.write(5 + 6)">Cliquez moi</button>
</body>
</html>
```

innerHTML

Écrit dans un élément HTML.

Nous allons cibler l'élément HTML par son ID.

```
<!DOCTYPE html>
<html>
<body>
     <h1>Titre de la page</h1>
     <p>Ceci est un paragraphe HTML</p>
     <p id="demo"></p>
     <script>
          document.getElementById("demo").innerHTML = 5 + 6;
     </script>
</body>
</html>
```

console.log()

Écrit une sortie dans la console du navigateur (F12 pour voir la console)

```
<!DOCTYPE html>
<html>
<body>
     <h1>Titre de la page</h1>
     <p>Ceci est un paragraphe HTML</p>
     <script>
          console.log(5 + 6);
     </script>
</body>
</html>
```

JavaScript

Les fonctions

Les fonctions

- Une fonction est une procédure, un ensemble d’instructions.
- on distingue deux types de fonctions :
  - les fonctions native, implanté à JavaScript.
  - les fonctions utilisateur, créée par l’utilisateur de JavaScript *(les développeurs)*

Les fonctions utilisateur

- Il est nécessaire de définir une fonction avant de l'utiliser.
- On définit une fonction avec
  - le mot clé `function`
  - suivi de son nom
  - une liste d'argument entre ( )
  - une série d'instructions entre { }

Les fonctions : Syntaxe

Déclaration de fonction :

```
function maFonction( argument1, argument2, … ) {
    instruction1;
    instruction2;
}
```

Expression de fonction :

```
var maFonction = function( argument1, argument2, … ) {
    instruction1;
    instruction2;
};
```

Les fonctions : utilisation

- l'**utilisation d'une fonction** s'appel aussi **appel d'une fonction**.
- Appel de la fonction, sans passage de paramètre :
  - `maFonction();`
- Appel de la fonction, avec passage de paramètres :
  - `maFonction( "Trillian" );`

Les fonctions : valeur de retour

- Une fonction peut retourner une valeur avec le mot clé `return`.

```
function cube( nombre ) {
    return nombre * nombre * nombre;
}
cube(2); // 8
```

Les fonctions : valeur par défaut

- Implanté avec ES6
- Les arguments peuvent avoir une valeur par défaut.

```
function multiplier( a, b=1 ) {
    return a * b;
}

multiplier(5); // 5
multiplier(5,2); // 10
```

JavaScript

Les exceptions

try … catch

- L'instruction `try … catch` regroupe des instructions à exécuter et définit une réponse si l'une de ces instructions provoque une exception.

```
try {
    instruction1;
    instruction2;
} catch(e) {
    console.log(e);
}
```

Throw

- L'instruction `throw` permet de lever une exception définie par l'utilisateur.

```
var month = 13;

function ExceptionMois(message) {
    this.message = message;
    this.name = "ExceptionMois";
}

if (month > 12) {
    throw new ExceptionMois("Il n’y à pas plus de 12 mois dans une
année.");
}
```

JavaScript

Les expressions régulière

Les expressions régulière

- Aussi appelées **expressions rationnelle**.
- Ce sont des motifs utilisés pour correspondre à certaines combinaisons de caractères au sein d’une chaîne.
- En JS les expressions rationnelle sont des objets.

Créer une expression rationnelle

- Avec un littéral d’expression régulière :
  - `var regex = /motif/drapeaux;`
- Avec le constructeur RegExp :
  - `var regex = new RegExp(“motif”, “drapeaux”);`

Expression rationnelle : drapeaux

- Les expressions rationnelle peuvent utiliser 4 drapeaux.
- Utilisés ensemble ou séparément.
- Utilisés dans n’importe quel ordre.

Expression rationnelle : drapeaux

- `g` : Recherche globale
- `i` : Recherche ne respectant pas la casse
- `m` : Recherche sur plusieurs lignes
- `u` : Motif considéré comme une séquence Unicode
- `y` : Recherche qui “adhère”, recherche à partir d’un index.

Expression rationnelle : motif

- Le motif est ce qu'on cherche à faire correspondre à l'expression rationnelle.
  - voir le fichier : Significations des caractères.pdf

```
var str = “Le chat à 4 pattes.”

var re = /chat/;

str.replace(re, “chien”); // Le chien à 4 pattes.
```

JavaScript

Math

L’objet Math

L’objet Math de JavaScript permet d’effectuer certaine tâches mathématiques sur les nombres.
Voici les méthodes les plus couramment utilisées :

- `Math.round()` arrondi à l’entier le plus proche.
  - `Math.round(4.7); // 5`
  - `Math.round(4.4); // 4`
- `Math.ceil()` retourne l’entier supérieur
  - `Math.ceil(8.2); // 9`
- `Math.floor()` retourne l’entier inférieur
  - `Math.ceil(8.7); // 8`

L’objet Math

- `Math.min()` retourne la valeur minimum de la série
  - `Math.min(0, 150, 30, 20, -8, -200); // -200`
- `Math.max()` retourne la valeur maximum de la série
  - `Math.max(0, 150, 30, 20, -8, -200); // 150`
- `Math.random()` retourne un chiffre aléatoire compris entre 0 et 1
  - `Math.random();`

JavaScript

JSON

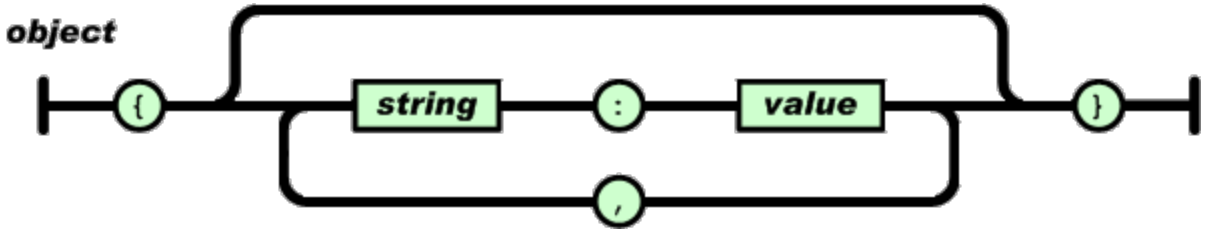
JSON

- **J**ava**S**cript **O**bject **N**otation
- Format léger d’échange de données.
- Facile à lire ou à écrire pour les humains.
- Facilement analysé et généré par des machine.
- Toute la documentation sur JSON : http://json.org/json-fr.html

JSON

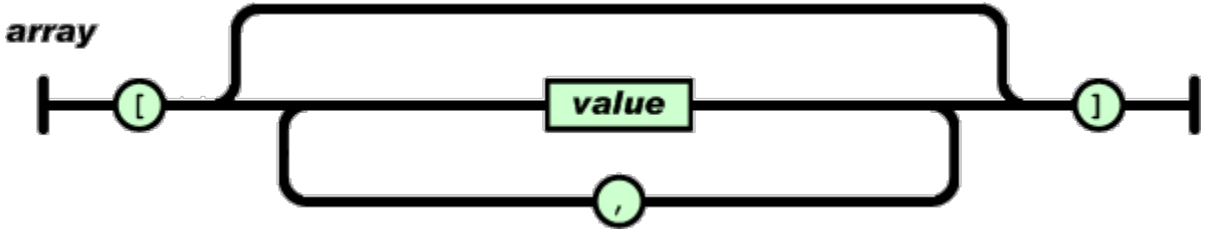
- Syntaxe :
  - `{ “username” : “Trillian” }`
- Basé sur deux structure :
  - une collection de paire nom:valeur
  - une liste de valeur ordonnées (un tableau)

JSON : Collection de paire nom:valeur




```
{
        “name” : “Trillian”,
        “age” : 32
}
```

JSON : Liste de valeurs ordonnées




```
{
        [
                “valeur”,
                “valeur”
    ]
}
```

JSON : Les types de valeur