

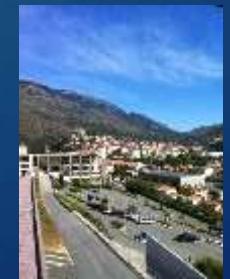
# Cours de programmation web

PAUL-ANTOINE BISGAMBIGLIA

2020-2021 / V2015-2016

[HTTP://PAUL-ANTOINE-BISGAMBIGLIA.UNIV-CORSE.FR/](http://PAUL-ANTOINE-BISGAMBIGLIA.UNIV-CORSE.FR/)

[@PABISGAMBIGLIA](mailto:@PABISGAMBIGLIA)



## Licence



- ▶ Cours en licence libre
- ▶ Certaines images ont été récupérées sur google image avec le filtre « réutilisation utilisée sans but commercial »
- ▶ Certains exemples de code sont issus de sites web référencés en fin de chapitre via leurs urls

# Objectifs

- ▶ Approfondir les notions de programmation web
- ▶ Apprendre à créer un site internet riche
- ▶ Technologies utilisées
  - ▶ HTML5
  - ▶ CSS3
  - ▶ JavaScript
  - ▶ PHP
  - ▶ MySQL
  - ▶ Des frameworks



# Prérequis

## Master 1

- ▶ Cours de L3
  - ▶ Algorithmique et programmation
  - ▶ Notion de réseau
    - ▶ Communication client serveur
    - ▶ Protocole HTTP
  - ▶ Techo web : HTML/CSS/JS/PHP

## Licence 3

- ▶ Notion d'algorithmique et de programmation
- ▶ Notion de réseau
  - ▶ Communication client serveur
  - ▶ Protocole HTTP

# Programme

- ▶ Introduction et révisions
  - ▶ Le web, et ses techno
- ▶ HTML 5
  - ▶ Langage de balise <body> </body>
- ▶ CSS 3 avec bootstrap
  - ▶ Mise en forme
  - ▶ Ergonomie du web
- ▶ JavaScript avec jQuery
  - ▶ Dynamisme coté client
  - ▶ Angular.js
  - ▶ Node.js
- ▶ PHP
  - ▶ Dynamisme coté serveur
  - ▶ Les frameworks (Zend, Symfony, cakephp)
- ▶ Le +
  - ▶ les Design Patterns pour le web
  - ▶ Sécuriser son code
  - ▶ Django le framework web en python
  - ▶ Framework html 5 pour mobile (IONIC, Mobile Angular UI, Intel XDK, Appcelerator Titanium, Sencha Touch)
  - ▶ Apache Cordova, Meteor

JavaScript  
est souvent  
abrégé en "JS"

1. LE LANGAGE
2. INTERACTIONS AVEC L'HTML  
ET LE CSS
3. BONNES PRATIQUES
4. LIMITES
5. LES FRAMEWORKS

# JS introduction

- ▶ JavaScript est un langage de script, multiplateformes et orienté objet.
- ▶ C'est un langage léger qui doit faire partie d'un environnement hôte pour qu'il puisse être utilisé sur les objets de cet environnement.
- ▶ JavaScript contient une bibliothèque standard d'objets tels que Array, Date, et Math, ainsi qu'un ensemble d'éléments de langage tels que les opérateurs, les structures de contrôles et les instructions.

# JS introduction

```
<!DOCTYPE html>
<html>
<body>
<h1> Un exemple JavaScript</h1>
<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Cliquer pour afficher la date
</button>
<p id="demo"></p>
</body>
</html>
```

- ▶ Le JS est un langage à objets utilisant le concept de **prototype**, disposant d'un **typage faible** et **dynamique** qui permet de programmer suivant plusieurs paradigmes de programmation :
  - ▶ fonctionnelle,
  - ▶ impérative et
  - ▶ orientée objet

# JS introduction

- ▶ Le standard pour JavaScript est ECMAScript.
- ▶ En **2012**, tous les navigateurs modernes supportent ECMAScript 5.1.
- ▶ Les anciens navigateurs supportent au minimum ECMAScript 3.
- ▶ Une sixième version majeure du standard a été publiée le 17 juin 2015. Elle s'intitule officiellement **ECMAScript 2015** mais est encore fréquemment appelée ECMAScript 6 ou ES6.
- ▶ les standards ECMAScript sont édités sur un rythme annuel, la dernière version est [ECMAScript 2020](#).

```
<!DOCTYPE html>
<html>
<body>
<h1>Exemple JS</h1>

<p>Cliquer sur l'image</p>
<script>
function changelImage() {
    var image = document.getElementById('myImage');
    if (image.src.match("bulbon")) {
        image.src = "pic_bulboff.gif";
    } else {
        image.src = "pic_bulbon.gif";
    }
}
</script>
</body>
</html>
```

# JS introduction

```
<!DOCTYPE html>
<html>
<body>
<h1> Exemple JS</h1> <p>saisir un nbr entre 1 et 10 :</p>
<input id="numb" type="number">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>
<script>
function myFunction() {
    var x, text;
    x = document.getElementById("numb").value;
    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}</script></body></html>
```

- ▶ JavaScript a été créé en 1995 par Brendan Eich, un ingénieur de Netscape
- ▶ Sorti avec Netscape 2 au début de l'année 1996
- ▶ 1996 Microsoft a lancé avec Internet Explorer 3 une version du langage globalement compatible, appelée Jscript
- ▶ 1997 normalisation [ECMAScript](#)

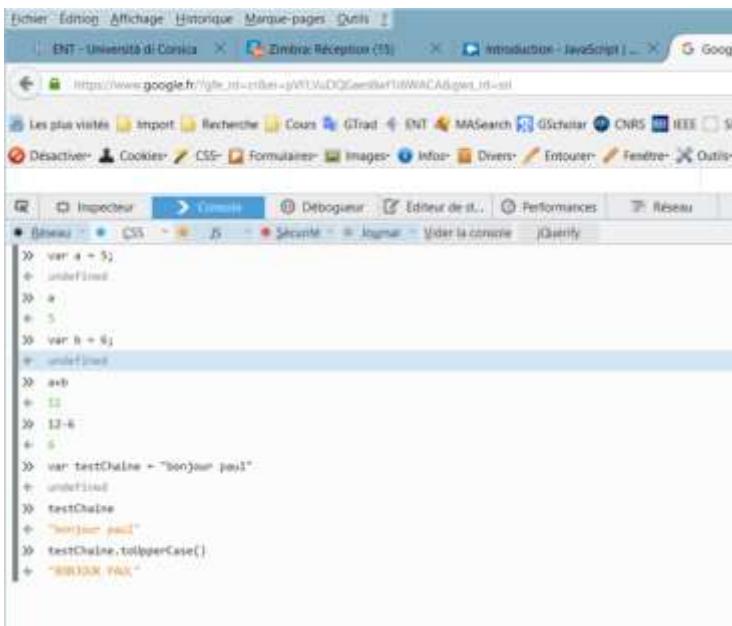
# JavaScript et Java

JavaScript et Java se ressemblent sur certains aspects mais ils sont fondamentalement différents :

- ▶ JS
  - ▶ typage faible
  - ▶ typage dynamique
  - ▶ Système interprété
  - ▶ types de données de base
  - ▶ modèle basé sur les prototypes pour représenter les liens entre les objets
- ▶ JAVA
  - ▶ typage fort
  - ▶ typage statique
  - ▶ système compilé
  - ▶ classes déclarées
  - ▶ modèle basé sur les classes

# Pour débuter

## Console web



```

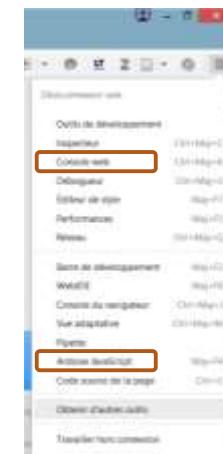
Dossier Édition Affichage Historique Marque-pages Outils 2
ENT - Université de Corse Zimbra Réception (5) Introduction à JavaScript Google

https://www.google.fr/?q=jd+créer+un+formulaire+avec+une+validation+JavaScript+et+HTML
Les plus visités Import Recherche Courriels GTrad ENT MASearch Gscholar CNRS IEEE SM
Déactiver Cookies CSS Formulaires Images Info Divers Fenêtre Outils

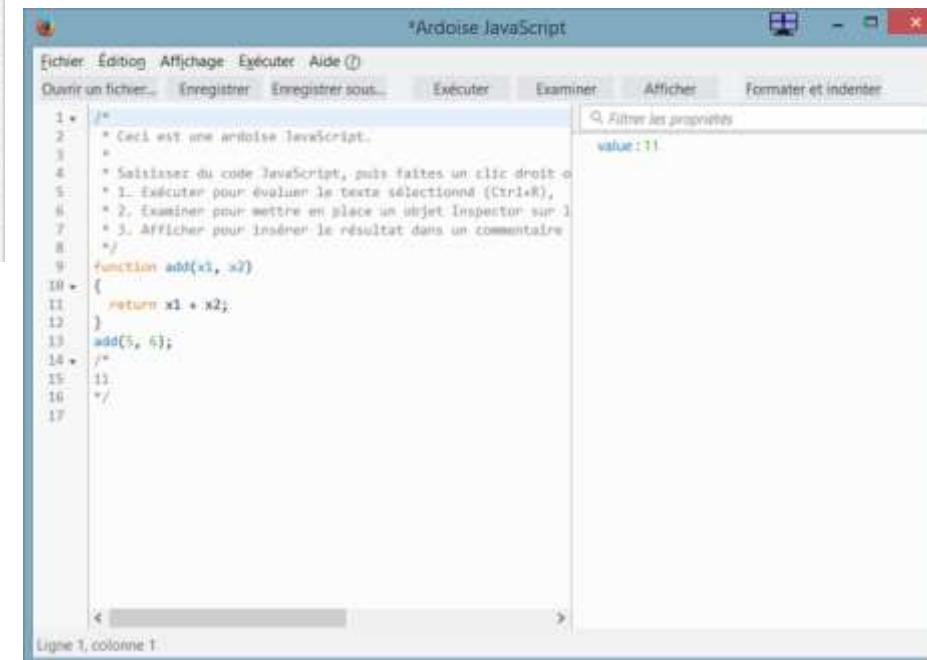
Fichier Édition Affichage Développeur Inspecteur Déboguer Éditeur de style Performances Réseau
Outils de développement
Inspecteur Ctrl+Shift+C Ctrl+Shift+K Ctrl+Shift+I
Déboguer
Éditeur de style
Performances
Réseau
Sous au développement
WAVES
Console du navigateur
Vue adaptative
Thème
Actions déclenchées
Code source de la page Ctrl+U
Obtenir d'autre code
Toujours non connecté

var a = 5;
a
5
var b = 6;
a+b
11
12-6
6
var testChaine = "bonjour paul";
testChaine
"bonjour paul"
testChaine.toUpperCase();
"BIJOUX PAUL"

```



## L'ardoise



Ardoise JavaScript

Fichier Édition Affichage Exécuter Aide ?

Ouvrir un fichier... Enregistrer Enregistrer sous... Exécuter Examiner Afficher Formater et indenter

```

1 /*
2 * Ceci est une ardoise JavaScript.
3 *
4 * Saisissez du code JavaScript, puis faites un clic droit sur
5 * 1. Exécuter pour évaluer le texte sélectionné (Ctrl+R),
6 * 2. Examiner pour mettre en place un objet Inspector sur l
7 * 3. Afficher pour insérer le résultat dans un commentaire
8 */
9 function add(x1, x2)
10 {
11     return x1 + x2;
12 }
13 add(5, 6);
14 /**
15 11
16 */
17

```

Filter les propriétés value : 11

Ligne 1, colonne 1

# Les bases du langage

- ▶ JavaScript emprunte la plupart des éléments de sa syntaxe à Java mais sa syntaxe est également influencée par Awk, Perl et Python.
- ▶ JavaScript est **sensible à la casse** et utilise l'ensemble de caractères **Unicode**
  - ▶ les instructions sont séparées par des points-virgules. Les espaces, les tabulations et les caractères de nouvelles lignes sont considérés comme des blancs
  - ▶ le texte d'un code source JavaScript est analysé de gauche à droite et est converti en une série d'unités lexicales, de caractères de contrôle, de fins de lignes, de commentaires et de blancs

# Les bases du langage

## Commentaires

```
// un commentaire sur une ligne
```

```
/* un commentaire plus  
long sur plusieurs lignes  
*/
```

```
/* Par contre on ne peut pas /* imbriquer  
des commentaires */ SyntaxError */
```

## Déclarations

### **var**

On déclare une variable, éventuellement en initialisant sa valeur.

### **let**

On déclare une variable dont la portée est celle du bloc courant, éventuellement en initialisant sa valeur.

### **const**

On déclare une constante nommée, accessible en lecture seule.

# Les bases du langage

## Variables

Les variables sont utilisées comme des noms symboliques désignant des valeurs utilisées dans votre code. Les noms des variables sont appelés identifiants

Un identifiant JavaScript doit commencer par une lettre, un tiret bas (\_) ou un symbole dollar (\$). Les caractères qui suivent peuvent être des chiffres (0 à 9)

## Déclarations

**var x = 42**

Cette syntaxe peut être utilisée pour déclarer des variables locales ou globales.

**x = 42**

Cela déclare une variable globale et ne peut être changé localement

**let y = 13**

Cette syntaxe peut être utilisée pour déclarer une variable dont la portée sera celle du bloc

# Les bases du langage

## Variables

Une variable déclarée grâce à l'instruction **var** ou **let** sans valeur initiale définie vaudra **undefined**.

Tenter d'accéder à une variable qui n'a pas été déclarée provoquera l'envoi d'une exception **ReferenceError**.

Il est possible d'utiliser **undefined** pour déterminer si une variable possède une valeur

## Code

```
var a;  
  
console.log("La valeur de a est " + a);  
// le log contient "La valeur de a est  
// undefined"  
  
console.log("La valeur de b est " + b);  
// signale une exception  
ReferenceError  
  
if (input === undefined) ...
```

# Les bases du langage

## Les portées de variables

une variable déclarée avec var en dehors des fonctions, elle est appelée **variable globale** elle est disponible pour tout le code contenu dans le document.  
une variable déclarée dans une fonction, est appelée variable locale car elle n'est disponible qu'au sein de cette fonction.

Avant ECMAScript 6, JavaScript ne définissait pas de portée pour une instruction de bloc

### Code

```
if (true) {  
    var x = 5;  
}  
  
console.log(x); // 5  
  
  
if (true) { let y = 5; } console.log(y);  
// ReferenceError: y is not defined
```

# Les bases du langage

## Types de données

- ▶ Six types de données primitifs :
  - ▶ Type **booléen** : true et false.
  - ▶ Type **nul** (null),
  - ▶ Un type pour les valeurs indéfinies (**undefined**).
  - ▶ Un type pour les nombres. 42 or 3.14159.
  - ▶ Un type pour les chaînes de caractères. "Coucou"
  - ▶ Un type pour les symboles,
- ▶ et un type pour les objets (Object)

## Code

```
var réponse = 42;  
réponse = "Hello"  
x = "La réponse est " + 42; // "La  
réponse est 42"  
"37" - 7; // 30  
"37" + 7; // "377"
```

# Les bases du langage

## Conversion

Si un nombre est représenté en mémoire par une chaîne de caractères, il y a des méthodes pour effectuer la bonne conversion :

**parseInt()**

**parseFloat()**

## L'opérateur + unaire

`+ "1.1" = 1.1 // fonctionne seulement avec le + unaire`

# Les bases du langage

## Les littéraux de tableaux

Si un tableau est créé en utilisant un littéral dans un script du plus haut niveau, JavaScript interprète le tableau chaque fois qu'il évalue l'expression contenant le littéral. De plus, un littéral utilisé dans une fonction est créé chaque fois que la fonction est appelée.

### Code

```
var cafés = ["Brésilien", "Colombien",  
"Kona"];  
  
var poisson = ["Clown", , "Chat"];  
  
var maListe = ['maison', , 'école', ];  
  
var maListe = [ , 'maison', , 'école'];  
  
var maListe = [undefined, 'maison',  
undefined, 'école'];
```

# Les bases du langage

## Les littéraux objets

Un littéral objet est une liste de zéro ou plusieurs paires de propriétés définies par des paires de noms/valeurs

Les dict en python

## Code

```
var monDico =  
{nom:"Richard",prenom:"Pierre",age:"11  
2"}  
  
console.log(monDico.nom);  
console.log(monDico["nom"]);  
  
var monDico =  
{nom:"Richard",prenoms:{first:"Pierre",se  
cond:"Paul"},age:"112"}
```

# Les bases du langage

## Caractères spéciaux en JavaScript

\0 Octet null  
\b Retour arrière  
\f Saut de page  
\n Nouvelle ligne  
\r Retour chariot  
\t Tabulation  
\v Tabulation verticale  
\' Apostrophe ou guillemet droit simple

\\" Guillemet droit double  
\\" Barre oblique inversée  
\XXX Le caractère dont l'encodage Latin-1 est spécifié grâce à, au plus, 3 chiffres octaux XXX entre 0 et 377. \251, par exemple représente la caractère copyright.  
\xXX Le caractère dont l'encodage Latin-1 est spécifié par deux chiffres hexadécimaux entre 00 et FF. Ainsi, \xA9 correspond à la séquence hexadécimale pour le caractère copyright.  
\uXXXX Le caractère Unicode spécifié par quatre chiffres hexadécimaux XXXX. Ainsi, \u00A9 correspondra à la séquence Unicode du symbole copyright. Voir Unicode escape sequences.

# Les bases du langage

## Caractères spéciaux en JavaScript

```
var chemin = "c:\\\\temp";
```

```
var str = "cette chaîne \
est cassée \
sur plusieurs \
lignes."
console.log(str); // cette chaîne est
cassée sur plusieurs lignes.
```

# Les bases du langage

## Les instructions conditionnelles

```
if (condition) {  
    instruction_1;  
} else {  
    instruction_2;  
}
```

```
if (condition_1) {  
    instruction_1;  
} else if (condition_2) {  
    instruction_2;  
} else if (condition_n) {  
    instruction_n;  
} else {  
    dernière_instruction;  
}
```

# Les bases du langage

## Les instructions conditionnelles

Lors d'un test, les valeurs suivantes seront considérées comme équivalentes à false :

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

```
function checkData(maChaîne) {  
    if (maChaîne.length == 3) {  
        return true;  
    } else {  
        alert("Veuillez saisir trois caractères. " +  
            maChaîne + " n'est pas valide.");  
        return false;  
    }  
}
```

# Les bases du langage

## Les instructions conditionnelles

```
switch (expression) {  
    case label_1:  
        instructions_1  
        [break];  
    case label_2:  
        instructions_2  
        [break];  
    ...  
    default:  
        instructions_par_defaut  
        [break];  
}
```

```
switch (fruit) {  
    case "Orange":  
        console.log("Les oranges sont à 60 centimes  
        le kilo.");  
        break;  
    case "Pomme":  
        console.log("Les pommes sont à 32  
        centimes le kilo.");  
        break;  
    default:  
        console.log("Désolé, nous n'avons pas de " +  
        fruittype + ".");  
}
```

# Les bases du langage

## Gérer les exceptions

Il est possible de lever des exceptions avec l'instruction **throw** et de les gérer (les intercepter) avec des instructions **try...catch**.

L'instruction **throw** est utilisée afin de signaler (*throw* en anglais) une exception. Lorsqu'on signale une exception, on définit une expression qui contient la valeur à renvoyer pour l'exception :

`throw expression;`

Il est possible d'utiliser n'importe quelle expression, sans restriction de type.

# Les bases du langage

## Gérer les exceptions

L'instruction `try...catch` permet de définir un bloc d'instructions qu'on essaye (try en anglais) d'exécuter, ainsi qu'une ou plusieurs instructions à utiliser en cas d'erreur lorsqu'une exception se produit. Si une exception est signalée, l'instruction `try...catch` permettra de l'« attraper » (catch en anglais) et de définir ce qui se passe dans ce cas.

```
function getNomMois(numMois) {  
    numMois = numMois - 1; // On décale de 1  
    car les indices du tableaux commencent à 0  
    var mois = ["Janvier", "Février", "Mars", "Avril"  
    , "Mai", "Juin", "Juillet",  
    "Août", "Septembre", "Octobre",  
    "Novembre", "Décembre"];  
    if (mois[numMois] != null) {  
        return mois[numMois];  
    } else {  
        throw "NuméroMoisInvalide"; // Ici on utilise  
        l'instruction throw  
    }  
}
```

# Les bases du langage

## Gérer les exceptions

On définit une fonction qui prend un nombre et renvoie le nom du mois correspondant à ce nombre. Si la valeur fournie n'est pas comprise entre 1 et 12, on signale une exception avec la valeur "NuméroMoisInvalide". Lorsque cette exception est gérée dans le bloc catch, la variable nomMois recevra la valeur "inconnu".

```
try { // les instructions à essayer si tout  
    // se passe bien  
    nomMois = getNomMois(maVarMois);  
    // La fonction peut renvoyer une  
    // exception  
} catch (e) {  
    nomMois = "inconnu";  
    gestionErreurLog(e); // on gère  
    // l'erreur avec une fonction  
}
```

# Les bases du langage

## Gérer les exceptions

Un bloc catch peut aussi être utilisé afin de gérer les exceptions pouvant être générées par les instructions du bloc try.

```
try {  
    throw "monException"; // on génère  
    une exception  
} catch (e) {  
    // les instructions utilisées pour gérer  
    les exceptions  
    enregistrerErreurs(e); // on passe  
    l'objet représentant l'exception à une  
    fonction utilisée pour gérer les erreurs  
}
```

# Les bases du langage

## Gérer les exceptions

Le bloc finally contient les instructions à exécuter après les blocs try et catch. Il est exécuté dans tous les cas, qu'il y ait une exception de produite ou non. Si une exception est signalée et qu'il n'y a pas de bloc catch pour la gérer, les instructions du bloc finally seront tout de même exécutées.

```
ouvrirFichier();
try {
    écrireFichier(données); // Une erreur
    peut se produire
} catch(e){
    gérerException(e); // On gère le cas où
    on a une exception
} finally {
    fermerFichier(); // On n'oublie jamais de
    fermer le flux.
}
```

# Les bases du langage

## Types d'exception

En JavaScript, n'importe quel objet peut être signalé comme une exception.

### A voir :

- ▶ Les exceptions [ECMAScript](#)
- ▶ [DOMException](#)
- ▶ [DOMError](#)
- ▶ Les objets [Error](#)
- ▶ [promesses](#)

# Les bases du langage

## Boucles et itération

Les boucles permettent de répéter des actions simplement et rapidement.

```
var pas;  
for (pas = 0; pas < 5; pas++) {  
    // Ceci sera exécuté 5 fois  
    // la variable "pas" ira de 0 à 4  
    console.log("Faire un pas vers l'est");  
}
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ continue
- ▶ for...in
- ▶ for...of

for ([expressionInitiale]; [condition];  
[expressionIncrément])  
instruction

# Exemple for

```
function quantité(selectObject) {  
    var qtéSélectionnée = 0;  
    for (var i = 0; i < selectObject.options.length;  
i++) {  
        if (selectObject.options[i].selected) {  
            qtéSélectionnée++;  
        }  
    }  
    return qtéSélectionnée;  
}  
  
var btn = document.getElementById("btn");  
btn.addEventListener("click", function(){  
    alert('Nombre d\'options choisies : ' +  
quantité(document.selectForm.typesMusique))  
});
```

```
<form name="selectForm">  
    <p>  
        <label for="typesMusique">Veuillez choisir des genres  
musicaux, puis cliquez :</label>  
        <select id="typesMusique" name="typesMusique"  
multiple="multiple">  
            <option selected="selected">R&B</option>  
            <option>Jazz</option>  
            <option>Blues</option>  
            <option>New Age</option>  
            <option>Classique</option>  
            <option>Opera</option>  
        </select>  
    </p>  
    <p><button id="btn" type="button">Combien sont  
sélectionnés ?</button></p>  
</form>
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ continue
- ▶ for...in
- ▶ for...of

```
do  
  instruction  
  while (condition);  
  
do {  
  i += 1;  
  console.log(i);  
} while (i < 5);
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ continue
- ▶ for...in
- ▶ for...of

while (condition)  
instruction

```
n = 0;  
x = 0;  
while (n < 3) {  
    n++;  
    x += n;  
}
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ continue
- ▶ for...in
- ▶ for...of

label:  
instruction

memoBoucle:  
while (memo == true) {  
    faireQQC();  
}

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ **break**
- ▶ continue
- ▶ for...in
- ▶ for...of

```
var x = 0;
var z = 0
labelAnnuleBoucle: while (true) {
    console.log("Boucle externe :" + x);
    x += 1;
    z = 1;
    while (true) {
        console.log("Boucle interne :" + z);
        z += 1;
        if (z === 10 && x === 10) {
            break labelAnnuleBoucle;
        } else if (z === 10) {
            break;
        }
    }
}
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ **continue**
- ▶ for...in
- ▶ for...of

```
i = 0;  
n = 0;  
while (i < 5) {  
    i++;  
    if (i == 3) {  
        continue;  
    }  
    n += i;  
}
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ **continue**
- ▶ for...in
- ▶ for...of

```
vérifierJ:  
while (i < 4) {  
    console.log(i + "<br>");  
    i += 1;  
vérifierJ:  
while (j > 4) {  
    console.log(j + "<br>");  
    j -= 1;  
    if ((j % 2) == 0) {  
        continue vérifierJ;  
    }  
    console.log(j + " est impair.<br>");  
}  
console.log("i = " + i + "<br>");  
console.log("j = " + j + "<br>");  
}
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ continue
- ▶ **for...in**
- ▶ for...of

```
for (variable in objet) {  
    instruction  
}
```

```
function afficherProps(obj, nomObj) {  
    var result = "";  
    for (var i in obj) {  
        result += nomObj + "." + i + " = " +  
obj[i] + "\n";  
    }  
    result += "\n";  
    return result;  
}
```

# Les bases du langage

## Boucles et itération

différentes boucles fournies par JavaScript :

- ▶ for
- ▶ do...while
- ▶ while
- ▶ label
- ▶ break
- ▶ continue
- ▶ for...in
- ▶ **for...of**

```
for (variable of objet) {  
    instruction  
}
```

```
let arr = [3, 5, 7];  
arr.toto = "coucou";
```

```
for (let i in arr) {  
    console.log(i); // affiche "0", "1", "2",  
    "toto" dans la console  
}
```

```
for (let i of arr) {  
    console.log(i); // affiche "3", "5", "7"  
    dans la console  
}
```

# Les bases du langage

## Fonctions

Une définition de est construite avec le mot-clé **function**, suivi par :

- ▶ Le nom de la fonction.
- ▶ Une liste d'arguments à passer à la fonction, entre parenthèses et séparés par des virgules.
- ▶ Les instructions JavaScript définissant la fonction, entre accolades, { }.

```
function carré(nombre) {
    return nombre * nombre;
}

function maFonction(monObjet) {
    monObjet.fabricant = "Toyota";
}
var mavoiture = {fabricant: "Honda",
    modèle: "Accord", année: 1998};
var x, y;
x = mavoiture.fabricant; // x aura la
// valeur "Honda"
maFonction(mavoiture);
y = mavoiture.fabricant; // y aura la
// valeur "Toyota"
```

# Les bases du langage

## Expressions de fonction

Les expressions de fonction sont pratiques lorsqu'il s'agit de passer une fonction comme argument d'une autre fonction.

```
var carré = function (nombre) { return  
nombre * nombre };  
var x = carré(4); //x reçoit la valeur 16
```

```
var factorielle = function fac(n) {  
return n<2 ? 1 : n*fac(n-1) };  
  
console.log(factorielle(3));
```

# Les bases du langage

## Expressions de fonction

Les expressions de fonction sont pratiques lorsqu'il s'agit de passer une fonction comme argument d'une autre fonction.

```
function map(f, a) {  
    var resultat = [], // Créer un nouveau tableau Array  
    i;  
    for (i = 0; i != a.length; i++)  
        resultat[i] = f(a[i]);  
    return resultat;  
}  
  
map(function(x) {return x * x * x}, [0, 1, 2, 5, 10]);
```

# Les bases du langage

## Portée d'une fonction

On ne peut pas accéder aux variables définies dans une fonction en dehors de cette fonction : ces variables n'existent que dans la portée de la fonction. En revanche, une fonction peut accéder aux différentes variables et fonctions qui appartiennent à la portée dans laquelle elle est définie

```
// Les variables suivantes sont globales
var num1 = 20,
    num2 = 3,
    nom = "Licorne";
// Cette fonction est définie dans la portée globale
function multiplier() {
    return num1 * num2;
}
multiplier(); // Renvoie 60

// Un exemple de fonction imbriquée
function getScore () {
    var num1 = 2,
        num2 = 3;
    function ajoute() {
        return nom + " a marqué " + (num1 + num2);
    }
    return ajoute();
}
getScore(); // Renvoie "Licorne a marqué 5"
```

# Les bases du langage

## Closures

La fonction interne aura accès aux variables et paramètres de la fonction parente mais pas l'inverse.  
On crée une **closure** lorsque la fonction interne est disponible en dehors de la fonction parente

```
var animal = function(nom) { // La fonction externe utilise un paramètre "nom"
    var getNom = function () {
        return nom; // La fonction interne accède à la variable "nom" de la fonction externe
    }
    return getNom; // Renvoie la fonction interne pour la rendre disponible en dehors de la portée de la fonction parente
}

monAnimal = animal("Licorne");

monAnimal(); // Renvoie "Licorne"
```

# Les bases du langage

```
(( ) => console.log('Hello world'))();
```

## Closures

```
function OuterFunction() {  
    var outerVariable = 100;  
    function InnerFunction() {  
        alert(outerVariable); }  
    return InnerFunction; }  
var innerFunc = OuterFunction();  
innerFunc(); // 100
```

## Call back

```
function greet(name, callback) {  
    console.log('Hi' + ' ' + name);  
    callback(); }  
function callMe() {  
    console.log('I am callback function'); }  
greet('Peter', callMe);  
//Hi Peter  
//I am callback function
```

# Les bases du langage

## L' objet arguments

Les arguments d'une fonction sont stocké dans un objet semblable à un tableau `arguments[i]` où `i` représente l'index ordinal de l'argument ; Le nombre total d'arguments est fourni grâce à `arguments.length`.

```
function monConcat(séparateur) {  
    var result = "", // on initialise la liste  
    i;  
    // on parcourt les arguments  
    for (i = 1; i < arguments.length; i++) {  
        result += arguments[i] +  
        séparateur;  
    }  
    return result;  
}  
  
monConcat(" ", "red", "orange",  
"blue");
```

# Les bases du langage

## Les paramètres par défaut

```
function multiplier(a, b) {  
    b = typeof b !== 'undefined' ? b : 1;  
  
    return a*b;  
}  
  
multiplier(5);  
// 5
```

```
function multiplier(a, b = 1) {  
    return a*b;  
}  
  
multiplier(5);  
// 5
```

# Les bases du langage

## A voir aussi

- ▶ Fonctions fléchées
- ▶ Fonctions prédéfinies

eval()  
isFinite()  
isFinite()  
parseFloat()  
parseInt()  
decodeURI()  
decodeURIComponent()  
encodeURI()  
encodeURIComponent()

# Les bases du langage

## Opérateurs d'affectation

<u>Affectation</u>	$x = y$	$x = y$
<u>Affectation après addition</u>	$x += y$	$x = x + y$
<u>Affectation après soustraction</u>	$x -= y$	$x = x - y$
<u>Affectation après multiplication</u>	$x *= y$	$x = x * y$
<u>Affectation après division</u>	$x /= y$	$x = x / y$
<u>Affectation du reste</u>	$x \% y$	$x = x \% y$
<u>Affectation après exponentiation</u>	$x **= y$	$x = x ** y$
<u>Affectation après décalage à gauche</u>	$x <= y$	$x = x << y$
<u>Affectation après décalage à droite</u>	$x >= y$	$x = x >> y$
<u>Affectation après décalage à droite non signé</u>	$x >>= y$	$x = x >>> y$
<u>Affectation après ET binaire</u>	$x &= y$	$x = x \& y$
<u>Affectation après OU exclusif binaire</u>	$x ^= y$	$x = x \wedge y$
<u>Affectation après OU binaire</u>	$x  = y$	$x = x   y$

# Les bases du langage

## Opérateurs de comparaison

<u>Égalité</u> (==)	Renvoie true si les opérandes sont égaux après conversion en valeurs de mêmes types.	3 == var1 "3" == var1 3 == '3'
<u>Inégalité</u> (!=)	Renvoie true si les opérandes sont différents.	var1 != 4 var2 != "3"
<u>Égalité stricte</u> (===)	Renvoie true si les opérandes sont égaux et de même type. Voir <a href="#">Object.is()</a> et <a href="#">égalité de type en JavaScript</a> .	3 === var1
<u>Inégalité stricte</u> (!==)	Renvoie true si les opérandes ne sont pas égaux ou s'ils ne sont pas de même type.	var1 !== "3" 3 !== '3'
<u>Supériorité stricte</u> (>)	Renvoie true si l'opérande gauche est supérieur (strictement) à l'opérande droit.	var2 > var1 "12" > 2
<u>Supériorité ou égalité</u> (>=)	Renvoie true si l'opérande gauche est supérieur ou égal à l'opérande droit.	var2 >= var1 var1 >= 3
<u>Infériorité stricte</u> (<)	Renvoie true si l'opérande gauche est inférieur (strictement) à l'opérande droit.	var1 < var2 "2" < "12"
<u>Infériorité ou égalité</u> (<=)	Renvoie true si l'opérande gauche est inférieur ou égal à l'opérande droit.	var1 <= var2 var2 <= 5

# Les bases du langage

## Opérateurs logiques

Et aussi

- ▶ Opérateur conditionnel ternaire
  - ▶ condition ? val1 : val2
  - ▶ var statut = (âge >= 18) ? "adulte" : "mineur";
- ▶ La virgule comme opérateur
  - ▶ for (var i = 0, j = 9; i <= j; i++, j--)
- ▶ Opérateurs unaires
  - ▶ Delete
  - ▶ Typeof
  - ▶ void

<u>ET logique (&amp;&amp;)</u>	expr1 && expr2	Renvoie expr1 s'il peut être converti à false, sinon renvoie expr2. Dans le cas où on utilise des opérandes booléens, && renvoie true si les deux opérandes valent true, false sinon.
<u>OU logique (   )</u>	expr1    expr2	Renvoie expr1 s'il peut être converti à true, sinon renvoie expr2. Dans le cas où on utilise des opérandes booléens,    renvoie true si l'un des opérandes vaut true, si les deux valent false, il renvoie false.
<u>NON logique (!)</u>	!expr	Renvoie false si son unique opérande peut être converti en true, sinon il renvoie true.

# Les bases du langage

## Opérateurs relationnels

Un opérateur relationnel compare ses opérandes et renvoie une valeur booléenne selon que le résultat de la comparaison est vrai ou faux.

**In** l'opérateur in renvoie true si la propriété indiquée fait partie de l'objet donné.

"PI" in Math; // renvoie true

**Instanceof** L'opérateur instanceof renvoie true si l'objet donné est du type spécifié.

# Les bases du langage

## Précédence des opérateurs

La précédence des opérateurs indique l'ordre dans lequel ils sont appliqués lors de l'évaluation d'une expression. L'utilisation de parenthèses permet de surcharger la relation de précédence.

Type d'opérateur	Opérateurs individuels
membre	. []
appel/création d'instance	() new
négation/incrémantation	! ~ - + ++ -- typeof void delete
multiplication/division	* / %
addition/soustraction	+ -
décalage binaire	<< >> >>>
relationnel	< <= > >= in instanceof
égalité	== != === !==
ET binaire	&
OU exclusif binaire	^
OU binaire	
ET logique	&&
OU logique	
conditionnel	? :
assignation	= += -= *= /= %= <<= >>= >>>= &= ^=  =
virgule	,

# Les bases du langage

## Expressions

Un expression correspond à une unité de code valide qui est résolue en une valeur.

- ▶ **Arithmétiques** : l'expression est évaluée en un nombre (par exemple 3.14159)
- ▶ **Textuelles** : l'expression est évaluée en une chaîne de caractères
- ▶ **Logiques** : l'expression est évaluée en true ou false
- ▶ **Primaires** : Les mots-clés basiques et les expressions générales en JavaScript
- ▶ Expressions **vers la gauche** : Les valeurs à gauche sont la cible d'une affectation

# Les bases du langage

## Expressions primaires

Ces expressions correspondent aux mots-clés et aux expressions générales en JavaScript.

Le mot-clé **this** permet de faire référence à l'objet courant. En général, on l'utilise au sein d'une méthode pour faire référence à l'objet qui a utilisé la méthode.

```
function valide(obj, valMin, valMax){  
    if ((obj.value < valMin) || (obj.value >  
        valMax))
```

```
        console.log("Valeur incorrecte !");  
}
```

```
<p>Entrez un nombre entre 18 et 99 :</p>  
<input type="text" nom="age" size=3  
onChange="valide(this, 18, 99);">
```

# Les bases du langage

## Expressions vers la gauche

L'opérateur **new** permet de créer une instance d'un objet défini par l'utilisateur ou d'un objet dont le type est un des types d'objets natifs.

Le mot-clé **super** est utilisé afin d'appeler des fonctions disponibles sur un objet parent. Il peut notamment être utilisé avec les classes pour appeler le constructeur parent.

```
var nomObjet = new typeObjet([param1,  
param2, ..., paramN]);
```

```
super([arguments]); // invoque le constructeur  
parent
```

```
super.functionParent([arguments]);
```

### // Opérateur de décomposition

```
function f(x, y, z) {}  
var args = [0, 1, 2];  
f(...args);
```

# Les bases du langage

## `const`

- ▶ La **déclaration const** permet de créer une constante nommée accessible uniquement en lecture.
- ▶ Cela ne signifie pas que la valeur contenue est **immutable**, uniquement que l'identifiant ne peut pas être réaffecté.
- ▶ Autrement dit la valeur d'une constante ne peut pas être modifiée par des réaffectations ultérieures. Une constante ne peut pas être déclarée à nouveau.
- ▶ Cette déclaration permet de créer une constante qui peut être globale ou locale pour la fonction dans laquelle elle a été déclarée. Les constantes font partie de la portée du bloc (comme les variables définies avec `let`)

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/const>

# Les bases du langage

## **const**

- ▶ Il est nécessaire d'initialiser une constante lors de sa déclaration.
- ▶ Attention, la déclaration **const** crée une référence en lecture seule vers une valeur.
- ▶ Cela ne signifie pas que la valeur référencée ne peut pas être modifiée ! Ainsi, si le contenu de la constante est un objet, l'objet lui-même pourra toujours être modifié.

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/const>

# Les tableaux

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
let size = sequence.length;
for (var i = 0; i < size; i++)
{
    console.log(sequence[i]);
}
```

- ▶ .split(',');
- ▶ .length
- ▶ .join(',');
- ▶ .toString();
- ▶ .push('elemt');
- ▶ .pop();
- ▶ .unshift('elemt');
- ▶ .shift();

# Les tableaux

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++)
{
    console.log(sequence[i]);
}
```

## Exercices, réalisez une fonction pour :

1. Doubler chaque élément du tableau
2. Renvoyer un tableau avec les éléments paires
3. Renvoyer un tableau avec les éléments impaires
4. Faire la sommes des éléments du tableau
5. Indiquez si un élément est paire

# Les tableaux

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++)
{
    console.log(sequence[i]);
}
```

```
let sequence = [1,1,2,3,5,8,13]
const soustableau = sequence.map(e => e * 2)

▶ soustableau
▶ Array(7) [ 2, 2, 4, 6, 10, 16, 26 ]
```

**Doubler chaque élément du tableau**

# Les tableaux

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++)
{
    console.log(sequence[i]);
}
```

Renvoyer un tableau avec les éléments paires / impaires

```
const soustableauinaire = sequence.filter(e => e % 2 ==0)
```

- ▶ soustableauinaire
- ▶ Array [ 2, 8 ]

```
const soustableauimpaire = sequence.filter(e => e % 2 ==0)
```

# Les tableaux

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++)
{
    console.log(sequence[i]);
}
```

## Faire la sommes des éléments du tableau

```
const sommetableau =
sequence.reduce((sum,e)=>sum+e);
```

- ▶ sommetableau
- ▶ 33

Ici sum est un accumulateur

```
reduce(function (accumulator, currentValue,
currentIndex) { /* ... */ })  
reduce(function (accumulator, currentValue)
{ /* ... */ }, initialValue)
```

# Les tableaux

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (var i = 0; i < sequence.length; i++)
{
    console.log(sequence[i]);
}
```

**Indiquez si un élément est paire**

```
const isPaire = sequence.filter(e => e % 2
!= 0).length != 0;
▶ isPaire
▶ true
const words = ['spray', 'limit', 'elite',
'exuberant', 'destruction', 'present'];
const result = words.filter(word =>
word.length > 6);
console.log(result);
// expected output: Array ["exuberant",
"destruction", "present"]
```

# Bilan

- ▶ **map** : mon tableau de sortie est différent mais de même taille
- ▶ **filter** : mon tableau de sortie est plus petit, il est filtré
- ▶ **reduce** : je renvoie une valeur

# Bilan

## reduce

- ▶ 

```
const getMax = (a, b) => Math.max(a, b);
```
- ▶ 

```
[1, 100].reduce(getMax, 50); // 100
```
- ▶ 

```
[50].reduce(getMax, 10); // 50
```

# Exercice

- ▶ Sois le tableau :

```
const football = [  
  {club: ''SCB'', joueur='SANTELLI', but:2},  
  {club: ''SCB'', joueur='MAGRI', but:1},  
  {club: '' HAC'', joueur='KITALA', but:3},  
  {club: ''SCB'', joueur='ROBIC', but:3},  
  {club: ''BORDEAUX'', joueur='MAJA', but:4},  
]
```

**Combien de but ont marqué (somme) les joueurs du SCB après avoir ajouté 1 but à tous les joueurs du tableau !**

# Exercice

- ▶ Sois le tableau :

```
const football = [  
  {club: "SCB", joueur:"SANTELLI", but:2},  
  {club: "SCB", joueur:"MAGRI", but:1},  
  {club: "HAC", joueur:"KITALA", but:3},  
  {club: "SCB", joueur:"ROBIC", but:3},  
  {club: "BORDEAUX", joueur:"MAJA", but:4}  
]
```

```
const somme = football  
  .filter(player => player.club == "SCB")  
  .map(player => player.but +=1)  
  .reduce((somme,sumGoal) => somme+sumGoal);
```

# Les bases du langage

## Les objets

L'objet **Number** possède certaines propriétés représentant les constantes numériques telles que : la valeur maximale représentable en JavaScript, une valeur spéciale pour dire que la valeur numérique n'est pas un nombre et l'infini.

```
var plusGrandNombre = Number.MAX_VALUE;  
var plusPetitNombre = Number.MIN_VALUE;  
var infini = Number.POSITIVE_INFINITY;  
var infiniNégatif = Number.NEGATIVE_INFINITY;  
var pasUnNombre = Number.NaN;
```

# Les bases du langage

## Les objets

L'objet natif **Math** possède des propriétés et des méthodes statiques pour représenter des constantes et des fonctions mathématiques.

Méthode	Description
<a href="#">abs()</a>	Valeur absolue
<a href="#">sin()</a> , <a href="#">cos()</a> , <a href="#">tan()</a>	Fonctions trigonométriques standards (les arguments sont exprimés en radians)
<a href="#">asin()</a> , <a href="#">acos()</a> , <a href="#">atan()</a> , <a href="#">atan2()</a>	Fonctions trigonométriques inverses (les valeurs renvoyées sont exprimées en radians)
<a href="#">sinh()</a> , <a href="#">cosh()</a> , <a href="#">tanh()</a>	Fonctions trigonométriques hyperboliques (les arguments sont exprimés en radians)
<a href="#">asinh()</a> , <a href="#">acosh()</a> , <a href="#">atanh()</a>	Fonctions trigonométriques hyperboliques inverses (les valeurs renvoyées sont exprimées en radians).
<a href="#">pow()</a> , <a href="#">exp()</a> , <a href="#">expm1()</a> , <a href="#">log10()</a> , <a href="#">log1p()</a> , <a href="#">log2()</a>	Fonctions exponentielles et logarithmiques
<a href="#">floor()</a> , <a href="#">ceil()</a>	Renvoie le plus petit/grand entier inférieur/supérieur ou égal à l'argument donné
<a href="#">min()</a> , <a href="#">max()</a>	Renvoie le plus petit (resp. grand) nombre d'une liste de nombres séparés par des virgules
<a href="#">random()</a>	Renvoie un nombre aléatoire compris entre 0 et 1
<a href="#">round()</a> , <a href="#">fround()</a> , <a href="#">trunc()</a>	Fonctions d'arrondis et de troncature
<a href="#">sqrt()</a> , <a href="#">cbrt()</a> , <a href="#">hypot()</a>	Racine carrée, cubique et racine carrée de la somme des carrés des arguments
<a href="#">sign()</a>	Renvoie le signe d'un nombre et indique si la valeur est négative, positive ou nulle
<a href="#">clz32()</a> , <a href="#">imul()</a>	Le nombre de zéros qui commencent un nombre sur 32 bits en représentation binaire. La résultat de la multiplication de deux arguments sur 32 bits effectuée comme en C.

# Les bases du langage

## Les objets

L'objet **Date** et ses méthodes permettent de manipuler des dates et des heures au sein d'une application.

```
var monObjetDate = new Date([paramètres]);
```

- ▶ Aucun paramètre : l'objet créé représentera la date et l'heure courante.
- ▶ Une chaîne de caractères représentant une date au format suivant : "jour, année heures:minutes:secondes". Par exemple var noël95 = new Date("December 25, 1995 13:30:00"). Si les valeurs pour les heures, minutes ou secondes sont absentes, elles vaudront 0 par défaut.
- ▶ Un ensemble de valeurs entières pour l'année, le mois et le jour : var noël95 = new Date(1995, 11, 25).
- ▶ Un ensemble de valeurs entières pour l'année, le mois, le jour, l'heure, les minutes et les secondes : var noël95 = new Date(1995, 11, 25, 9, 30, 0);

# Les bases du langage

## Les objets

L'objet **Date** et ses méthodes permettent de manipuler des dates et des heures au sein d'une application.

```
var aujourd'hui = new Date();
// On définit le jour et le mois
var finAnnée = new Date(1995, 11, 31, 23, 59, 59, 999);
// On définit l'année avec l'année courante
finAnnée.setFullYear(aujourd'hui.getFullYear());
// On calcule le nombre de millisecondes par jour
var msParJour = 24 * 60 * 60 * 1000;

// On renvoie le nombre de jours restants dans l'année
var joursRestants = (finAnnée.getTime() -
today.getTime()) / msPerDay;
joursRestants = Math.round(joursRestants);
```

# Les bases du langage

## Les objets

L'objet **Date** et ses méthodes permettent de manipuler des dates et des heures au sein d'une application.

```
function JSClock() {  
    var temps = new Date();  
    var heures = temps.getHours();  
    var minutes = temps.getMinutes();  
    var secondes = temps.getSeconds();  
    var calc = "" + (heures > 12) ? heures - 12 : heures;  
    if (heures == 0)  
        calc = "12";  
    calc += ((minutes < 10) ? ":0" : ":") + minutes;  
    calc += ((secondes < 10) ? ":0" : ":") + secondes;  
    calc += (heures >= 12) ? " P.M." : " A.M.";  
    return calc;  
}
```

# Les bases du langage

## Les objets

L'objet **String** est une « enveloppe » (wrapper) objet autour du type primitif pour les chaînes de caractères.

```
var s1 = "toto"; // crée une valeur primitive qui  
est une chaîne
```

```
var s2 = new String("toto"); // crée un objet  
String
```

```
console.log(s1); // Affiche "toto" dans la  
console
```

```
typeof s1; // Renvoie "string"
```

```
console.log(s2); // Affiche String ["t", "o", "t", "o"]
```

```
typeof s2; // Renvoie 'object'
```

# Les bases du langage

## Les objets

L'objet **String** est une « enveloppe » (wrapper) objet autour du type primitif pour les chaînes de caractères.

Méthode	Description
<a href="#">charAt</a> , <a href="#">charCodeAt</a> , <a href="#">codePointAt</a>	Renvoie le caractère ou le code du caractère pour une position donnée.
<a href="#">indexOf</a> , <a href="#">lastIndexOf</a>	Renvoie la position d'une sous-chaîne donnée dans la chaîne ou la dernière position d'une sous-chaîne donnée.
<a href="#">startsWith</a> , <a href="#">endsWith</a> , <a href="#">includes</a>	Renvoie si oui ou non la chaîne courant commence, finit ou contient une chaîne donnée.
<a href="#">concat</a>	Combine le texte de deux chaînes de caractères et renvoie une nouvelle chaîne de caractères.
<a href="#">fromCharCode</a> , <a href="#">fromCodePoint</a> ,	Construit une chaîne de caractères à partir de la séquence de valeurs Unicode fournie. Cette méthode est une méthode statique qui s'applique au type String et non à une instance.

# Les bases du langage

## Les objets

L'objet **String** est une « enveloppe » (wrapper) objet autour du type primitif pour les chaînes de caractères.

<u>split</u>	Découpe un objet String en un tableau de chaînes de caractères qui sont des sous chaînes de la chaîne courante (selon un séparateur donné).
<u>slice</u>	Retire un fragment dans la chaîne et renvoie une nouvelle chaîne de caractères sans ce fragment.
<u>substring</u> , <u>substr</u>	Renvoie une sous-chaîne donnée de la chaîne courante, soit à partir d'une position de début et d'une position de fin ou à partir d'une position de début et d'une longueur.
<u>match</u> , <u>replace</u> , <u>search</u>	Fonctionne avec les expressions rationnelles.

# Les bases du langage

## Les objets

L'objet **String** est une « enveloppe » (wrapper) objet autour du type primitif pour les chaînes de caractères.

<a href="#"><u>toLowerCase</u></a> , <a href="#"><u>toUpperCase</u></a>	Renvoie la chaîne en minuscules ou en majuscules.
<a href="#"><u>normalize</u></a>	Renvoie la forme Unicode normalisée de la chaîne courante.
<a href="#"><u>repeat</u></a>	Renvoie une chaîne qui est la répétition d'un objet donné un certain nombre de fois.
<a href="#"><u>trim</u></a>	Retire les blancs en début et en fins de chaîne.

# Les bases du langage

## Les objets

L'objet **DateTimeFormat** peut être utilisé afin de mettre en forme les dates et heures.

L'objet **NumberFormat** permet de mettre en forme les valeurs numériques et notamment les devises

- ▶ L'objet **Collator** peut être utilisé pour comparer et ordonner des chaînes de caractères.
- ▶ L'objet et/ou le type **Arrays** permet de gérer des tableaux :
  - ▶ `var array-name = [item1, item2, ...];`
  - ▶ `var cars = ["Saab", "Volvo", "BMW"];`
  - ▶ `var name = cars[0];`
  - ▶ `cars.length;`
  - ▶ `cars.sort();`
  - ▶ `cars.push("Toyota"); or cars[id] = "Toyota"`

# Les bases du langage

## Mes objets

On peut créer des objets avec une fonction qui est un constructeur mais on peut aussi créer des objets avec des initialisateurs d'objets. On appelle parfois cette syntaxe la notation littérale.

```
var obj = { propriété_1: valeur_1, //  
propriété_# peut être un identifiant  
    2:      valeur_2, // ou un nombre  
    // ....  
    "propriété n": valeur_n }; // ou une  
chaîne
```

# Les bases du langage

## Mes objets

On peut créer des objets avec une fonction qui est un constructeur mais on peut aussi créer des objets avec des initialisateurs d'objets. On appelle parfois cette syntaxe la notation littérale.

```
function Voiture(fabricant, modèle, année) {  
    this.fabricant = fabricant;  
    this.modèle = modèle;  
    this.année = année;  
}  
  
var maVoiture = new Voiture("Eagle", "Talon  
TSi", 1993);
```

# Les bases du langage

## Mes objets

Les objets peuvent également être créés en utilisant la méthode `Object.create()`. Cette méthode peut s'avérer très utile car elle permet de choisir le prototype pour l'objet qu'on souhaite créer, sans avoir à définir un constructeur.

// Propriétés pour animal et encapsulation des méthodes

```
var Animal = {  
    type: "Invertébrés", // Valeur par défaut  
    afficherType : function(){  
        console.log(this.type);  
    }  
}
```

// On crée un nouveau type d'animal, animal1

```
var animal1 = Object.create(Animal);
```

```
animal1.afficherType(); // affichera Invertébrés
```

// On crée un type d'animal "Poissons"

```
var poisson = Object.create(Animal);
```

```
poisson.type = "Poisson";
```

```
poisson.afficherType(); // affichera Poissons
```

# Les bases du langage

## Mes objets

Comment parcourir les propriétés d'un objet

À partir d'ECMAScript 5, il existe trois méthodes natives pour lister/parcourir les propriétés d'un objet :

- ▶ Les boucles **for...in** qui permettent de parcourir l'ensemble des propriétés énumérables d'un objet et de sa chaîne de prototypes.
- ▶ **Object.keys(o)** qui permet de renvoyer un tableau contenant les noms (clés ou keys) des propriétés propres (celles qui ne sont pas héritées via la chaîne de prototypes) d'un objet o pour les propriétés énumérables.
- ▶ **Object.getOwnPropertyNames(o)** qui permet de renvoyer un tableau contenant les noms des propriétés propres (énumérables ou non) d'un objet o.

# Les bases du langage

## Mes objets

Définir des accesseurs et des mutateurs (getters et setters).

Un accesseur (getter) est une méthode qui permet de récupérer la valeur d'une propriété donnée. Un mutateur (setter) est une méthode qui permet de définir la valeur d'une propriété donnée.

```
var o = {
    a: 7,
    get b() {
        return this.a + 1;
    },
    set c(x) {
        this.a = x / 2
    }
};
```

# Les bases du langage

## Mes objets

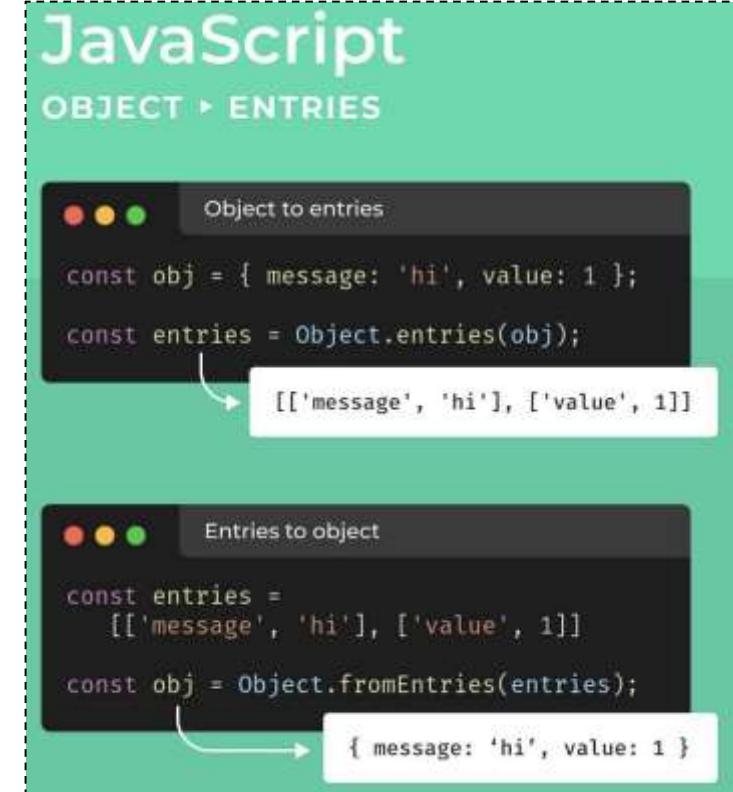
Source :  
[https://www.instagram.com/jscripter\\_s/](https://www.instagram.com/jscripter_s/)

```
function Student() {  
    this.name = 'John';  
    this.gender = 'M'; }  
Student.prototype.age = 15;  
var studObj1 = new Student();  
alert(studObj1.age); // 15  
var studObj2 = new Student();  
alert(studObj2.age); // 15
```

# Les bases du langage

## Transformation d'Objet

Source :  
[https://www.instagram.com/baby\\_wolf\\_codes/](https://www.instagram.com/baby_wolf_codes/)



# Les bases du langage

## Héritage

Tous les objets JavaScript héritent d'un autre objet. L'objet dont on hérite est appelé prototype et les propriétés héritées peuvent être accédées via l'objet prototype du constructeur

```
Voiture.prototype.couleur = null;  
voiture1.couleur = "noir";
```

# Les bases du langage

## Expression rationnelle

Les expressions rationnelles sont des motifs utilisés pour correspondre à certaines combinaisons de caractères au sein de chaînes de caractères. En JavaScript, les expressions rationnelles sont également des objets.

- ▶ Ces objets sont utilisés avec les méthodes **exec** et **test** de RegExp, et **match**, **replace**, **search** et **split** de String.
- ▶ Création
  - ▶ `var re = /ab+c/;`
  - ▶ `var re = new RegExp("ab+c");`
- ▶ Exemple `/ab*c/`
- ▶ correspond à toutes les combinaisons de caractères qui possèdent un seul 'a' suivi de zéro ou plusieurs 'b' (l'astérisque utilisée ici signifie que l'élément qui la précède doit être présent zéro ou plusieurs fois) qui sont immédiatement suivis d'un 'c'.

# Les bases du langage

## Expression rationnelle

- \ Un backslash précédant un caractère spécial indique que le caractère qui suit doit être interprété littéralement (et non pas comme un caractère spécial).
- ^ Correspond au début la séquence
- \$ Correspond à la fin de la séquence

- \* Correspond à l'expression précédente qui est répétée 0 ou plusieurs fois
- + Correspond à l'expression précédente qui est répétée une ou plusieurs fois
- ? Correspond à l'expression précédente qui est présente une fois ou pas du tout
- . (Le point) correspond à n'importe quel caractère excepté un caractère de saut de ligne
- {n} Correspond pour exactement n occurrences de l'expression précédente.

# Les bases du langage

## Et aussi

À partir d'ECMAScript 6, JavaScript fournit les objets natifs Proxy et Reflect. Ces objets permettent d'intercepter et de définir des comportements spécifiques pour certaines opérations fondamentales du langage (par exemple la recherche d'une propriété, l'affectation, l'énumération, l'appel d'une fonction, etc.). Grâce à ces deux objets, il est possible d'interagir avec le langage lui-même (on parle alors de métaprogrammation).

▶ À lire ici

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/M%C3%A9aprogrammation>

# Les bases du langage

## Et aussi

À partir d'ECMAScript 5, JavaScript propose un mode strict :

```
"use strict";
```

Ce mode permet d'écrire du code plus sécurisé, l'interpréteur ferra remonter toutes les erreurs.

```
"use strict";
```

```
x = 3.14;
```

► À lire ici

[http://www.w3schools.com/js/js\\_strict.asp](http://www.w3schools.com/js/js_strict.asp)

# Les bases du langage

## Et aussi

Mode debug !

Source

<https://www.instagram.com/reactjs1>

Example:

```
const spacing = '10px';
const styles = `padding: ${spacing}; background-color:
white; color: red; font-style: italic; border: 1px solid black;
font-size: 2em;`;
console.log('%cI am a styled log', styles);
```



À tester :

```
console.error('mon erreur');
```

```
console.warn('mon warning');
```

```
console.table({a:5,B:6});
```

Example:

```
console.group('group1');
console.warn('warning');
console.error('error');
console.log('I belong to a group');
console.groupEnd('group1');
console.log('I dont belong to any group');
```



# Les bases du langage

## JS et HTML

Balise <script> interne ou externe  
<script src="myScript.js"></script>

Fonctions et événements

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

<p>Fonction JS pour interagir avec le HTML</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Hello world.";
</script>

</body>
</html>
```

# Les bases du langage

## JS et HTML

### Affichage

- ▶ `window.alert()`
- ▶ `document.write()`
- ▶ `innerHTML`
- ▶ `console.log()`

```
<script>
window.alert(5 + 6);
document.write(5 + 6);
console.log(5 + 6);
document.getElementById("demo").innerHTML
= 5 + 6;
</script>
```

```
<p id="demo"></p>
<button onclick="document.write(5 + 6)">Try
it</button>
```

# Les bases du langage

## JS et HTML

- ▶ La validation de formulaire peut (doit aussi) être effectué en JS.
  - ▶ [http://www.w3schools.com/js/js\\_validation\\_api.asp](http://www.w3schools.com/js/js_validation_api.asp)
  - ▶ [http://www.w3schools.com/js/js\\_validation.asp](http://www.w3schools.com/js/js_validation.asp)

```
function validateForm() {  
    var x =  
    document.forms["myForm"]["fname"].value;  
    if (x == null || x == "") {  
        alert("Name must be filled out");  
        return false;  
    }  
}  
  
<form name="myForm" action="..."  
onsubmit="return validateForm()" method="post">  
Name: <input type="text" name="fname">  
<input type="submit" value="Submit">  
</form>
```

# Les bases du langage

## Events

Les événements HTML sont des "choses" qui surviennent d'éléments HTML.

Lorsque le JavaScript est utilisé dans des pages HTML, il peut «réagir» à ces événements.

Event	Description
onchange	Un élément HTML a changé
onclick	L'utilisateur clique sur l'élément
onmouseover	L'utilisateur bouge son curseur sur l'élément
onmouseout	L'utilisateur déplace la souris en dehors d'un élément HTML
onkeydown	L'utilisateur appuie sur une touche du clavier
onload	Le navigateur a fini de charger la page

# Les bases du langage

## Debugger

- ▶ Il est difficile de corriger du code JavaScript sans un débogueur !
- ▶ Outils ;
  - ▶ **console.log()**
  - ▶ **debugger**

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p id="demo"></p>
<p>En mode debug le script est arrêté ligne  
2.</p>
<script>
var x = 15 * 5;
debugger;
document.getElementById("demo").innerHTML =
x;
</script>
</body>
</html>
```

# Les bases du langage

## Debugger

- ▶ Chrome
  - ▶ Menu>developer tools>Console
- ▶ Firefox install Firebug
- ▶ IE
  - ▶ Menu>tools>developer tools>Console
- ▶ Opera
  - ▶ <http://dev.opera.com>
- ▶ Safari install Firebug Lite

### Aussi !

IROH.JS : un est outil d'analyse dynamique de code pour JavaScript

L'analyse statique du code permet de détecter les erreurs de codage, mais cela n'assure pas pour autant que votre code se comporte comme vous l'avez espéré. Si ce n'est pas le cas, l'analyse dynamique de code peut vous aider.

Il patche votre code afin d'enregistrer tout ce qui se passe au sein de celui-ci. Il garde une trace de la pile d'appel pour vous permettre de visualiser comment votre code se comporte.

# Les + du langage

## Promises et fonctions fléchées

- ▶ [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets\\_globaux/Promise](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise)

([param] [, param]) => { instructions } (param1, param2, ..., param2) => expression // équivalent à (param1, param2, ..., param2) => { return expression; }

# Les + du langage

## Promises et fonctions fléchées

- ▶ Source :  
<https://www.instagram.com/reactjs1/>

**A Promise has three states:**

**fulfilled: Action related to the promise succeeded**

**pending: Promise is still pending i.e not fulfilled or rejected yet**

**rejected: Action related to the promise failed**

# Les + du langage

## Promises et fonctions fléchées

- ▶ Source :  
<https://www.instagram.com/reactjs1/>

### Create Promises

```
const isGoodGrade = true;  
  
// Promise  
const willIGetNewPhone = new Promise(  
  (resolve, reject) => { // fat arrow  
    if (isGoodGrade) {  
      const phone = {  
        brand: "Samsung",  
        color: "black"  
      };  
      resolve(phone);  
    } else {  
      const reason = new Error("Did not get good grade.");  
      reject(reason);  
    }  
  }  
);
```

# Les + du langage

## Promises et fonctions fléchées

- ▶ Source :  
<https://www.instagram.com/reactjs1/>

### Consuming Promises

```
// call our promise
var askMom = function () {
  willIGetNewPhone
    .then(function (fulfilled) {
      // yay, you got a new phone
      console.log(fulfilled);
      // output: { brand: "Samsung", color: "black" }
    })
    .catch(function (error) {
      // oops, mom dont buy it
      console.log(error.message);
      // output: "Did not get good grade"
    });
};

askMom();
```

# Les + du langage

## Promises et fonction asynchrone

- ▶ Source :  
[https://www.instagram.com/baby\\_wolf\\_codes/](https://www.instagram.com/baby_wolf_codes/)

```
const displayData = async () => {
  const data = await fetch('https://api.github.com/repositories');
  const jsonData = await data.json();
  console.log(jsonData);
};
displayData();
```

<https://www.instagram.com/jscripters/>

```
function sleep(ms) {
  return new Promise(resolve => {
    setTimeout(resolve, ms)
  });
}
```

### USAGE

```
async function someFunction() {
  // Logic
  await sleep(2000);
  // More logic
}
```

# Les + du langage

## Promises et fetch

- ▶ Source :  
<https://www.instagram.com/codingkites/>



A screenshot of a browser developer tools console window. The title bar says "JS". The code shown is:

```
fetch("https://example.com/jsonfile.json")
  .then(response => response.json())
  .then(data => console.log(data));
```

An arrow points from the "data" variable in the code to the JSON object displayed below.

```
{ id: 1,
  name: "ABC",
  url: "abc.com",
  lang: ['JS', 'CSS'] }
```

# Les + du langage

## Promises et fetch

- Source :

<https://www.instagram.com/codingkites/>

**response.text()** - fetch with single argument.

**response.json()** - parse the response as JSON

**response.formData()**  
return the response as **FormData** object

**response.blob()**  
return the response as blob.

**response.arrayBuffer()**  
return the response as **ArrayBuffer**

```
let promise = fetch("url", [options])
    .then();
```

```
{
  method: 'POST',
  headers: {
    'Content-Type': '...'
}
```

# Règles Générales

- ▶ L'usage de variables globales doit être évité
- ▶ Les variables locales doivent être déclarées à l'aide du mot clé var
- ▶ La base doit toujours être spécifiée lors de l'usage de parseInt
- ▶ Le mode strict doit être utilisé
- ▶ L'usage de eval() doit être évité
- ▶ Un analyseur de code devrait être utilisé pendant le développement

## ▶ A lire aussi

- ▶ <http://www.js-attitude.fr/2013/01/21/dix-bonnes-pratiques-javascript/>
- ▶ <http://maxlab.fr/2014/12/bonnes-pratiques-javascript-pour-lentreprise/>
- ▶ [http://www.w3schools.com/js/js\\_best\\_practices.asp](http://www.w3schools.com/js/js_best_practices.asp)
- ▶ [http://www.w3.org/wiki/JavaScript\\_best\\_practices](http://www.w3.org/wiki/JavaScript_best_practices)
- ▶ <http://modernweb.com/2013/12/23/45-useful-javascript-tips-tricks-and-best-practices/>
- ▶ <https://zestedesavoir.com/tutoriels/358/module-pattern-en-javascript/>

# Bonnes pratiques

## Portée des variables

```
function maFonction(){  
    var a = 10;  
  
    //a est accessible ICI  
  
}  
  
//a n'est pas accessible ICI
```

```
function maFonction(){  
    a = 10;  
  
    //ce qu'il ne faut PAS FAIRE !  
}  
  
maFonction();  
console.log(a); // affichera 10 !! a est  
global
```

# Bonnes pratiques

## Fonctions dans des fonctions

```
function maFonction(){  
    //création d'une fonction à portée privée  
    function test(){  
        console.log('Je suis une fonction contenue  
dans maFonction');  
    }  
    b();  
}  
maFonction();
```

## Fonctions anonymes

```
var bonjour = function(){  
    console.log("Je suis une fonction  
anonyme");  
};  
bonjour();  
//affichera 'Je suis une fonction  
anonyme' dans la console
```

# Règles Générales

- ▶ Les fichiers et/ou principales fonctionnalités d'un programme devraient être enrobées dans un module
- ▶ Le pattern module est une manière d'encapsuler du code dans un package ou namespace tout en permettant si besoin, un accès extérieur à certaines propriétés/fonctions.

```
// Cette exemple montre l'utilisation d'une IIFE pour
réaliser un module
var monModule = (function (jQuery) {
    function methodePrivee(){
        jQuery(".content").html("hello");
    }
    function methodePrivee2(){
        console.log("hello");
    }
    return{
        methodePublique: function(){
            methodePrivee();
        }
    };
    // Exemple d'import d'un objet
})(jQuery);
monModule.methodePublique();
```

# Règles de syntaxe

- ▶ La notation littérale doit être utilisée pour créer des objets, tableaux, expressions régulières et primitives
- ▶ Par défaut, l'égalité strict (==) doit être utilisée

```
// écrire  
var monObjet = {} ;  
var monTableau = [] ;
```

```
// Plutôt que  
var monObjet = new Object() ;  
var monTableau = new Array() ;
```

```
var a1 = ('1' == 1); // true  
  
var a2 = ('1' === 1); // false  
  
var a3 = ('true' == true); // true  
  
var a4 = ('true' === true); // false
```

```
const odd = [1,3,5];  
const combined = [2,4,6, ...odd];  
console.log(combined);  
// [ 2, 4, 6, 1, 3, 5 ]
```

<https://www.instagram.com/jscripters/>

# Règles de syntaxe

- ▶ Une expression doit toujours se terminer par un point-virgule
- ▶ Une condition doit toujours utiliser des accolades
- ▶ Utilisez les paramètres par défaut dans vos fonctions

```
function myFunction(x, y) {  
    if (y === undefined) {  
        y = 0;  
    }  
}
```

## A lire

- ▶ <http://www.js-attitude.fr/2013/01/21/dix-bonnes-pratiques-javascript/>
- ▶ <http://www.js-attitude.fr/2012/12/26/convertir-un-nombre-en-texte-en-javascript/>
- ▶ <http://modernweb.com/2013/12/23/45-useful-javascript-tips-tricks-and-best-practices/>
- ▶ [http://www.w3schools.com/js/js\\_best\\_practices.asp](http://www.w3schools.com/js/js_best_practices.asp)

# Règles appliquées au javascript côté navigateur

- ▶ Les scripts javascript doivent être placés dans des fichiers externes
- ▶ Les scripts javascript doivent être appelés de préférence en fin de page
- ▶ Les sélecteurs doivent être mis en cache lors d'un usage répété
- ▶ Lorsque cela est possible, les fonctions natives du DOM doivent être utilisées
- ▶ La fonction document.write() ne devrait pas être utilisée

```
// Mauvais
for (var i = 0; i < 10000; i++) {
    $('#container').html(i);
}

// Bon
var monElement = $('#container')
for (var i = 0; i < 10000; i++) {
    monElement.html(i);
}

//Meilleur ( usage d'un sélecteur natif )
var monElement =
document.getElementById(container)
for (var i = 0; i < 10000; i++) {
    monElement.innerHTML = i;
}
```

# Règles appliquées au javascript côté navigateur

- ▶ Le code javascript devrait être séparé des balises HTML
  - ▶ Description : Afin de ne pas mélanger les langages , préférer l'usage de sélecteurs pour affecter des événements ou interagir avec des éléments du DOM.
  - ▶ Justification : Meilleur maintenabilité du code.
  - ▶ Commentaire : ceci est aussi valable pour l'attribut style

```
// Appel de fonction directement dans le HTML ( inline)
<div id="monElement" onclick="mafonction()">

// Depuis le code Javascript, affectation d'une fonction sur
// un attribut d'un élément sélectionné
var monElement =
document.getElementById("monElement");

monElement.onclick=function(){...};

// Utilisation du gestionnaire d'événement ( permet l'ajout
// de plusieurs écouteurs sur un même événement)
monElement.addEventListener("click", function() {
...
});
```

# Optimisation des performances

## Optimisation de boucle

- ▶ Le mauvais code accède à la longueur d'un tableau à chaque tour de boucle.
- ▶ Le meilleur code accède à la longueur en dehors de la boucle, et est donc plus rapide.

**for (i = 0; i < arr.length; i++) {**

**NO**

**I = arr.length;**

**for (i = 0; i < I; i++) {**

**YES**

# Optimisation des performances

## Optimisation d'accès au DOM

- ▶ L'accès au DOM est très lent, comparé aux autres instructions JavaScript.
- ▶ Si vous devez accéder à un élément du DOM à plusieurs reprises, utilisez une variable locale

```
obj = document.getElementById("demo");
obj.innerHTML = "Hello";
```

# Optimisation des performances

## Optimisation au DOM

- ▶ Ne pas créer de variables pour rien

```
var fullName = firstName + " " + lastName;  
document.getElementById("demo").innerHTML = fullName;
```

**NO**

```
document.getElementById("demo").innerHTML =  
firstName + " " + lastName
```

**YES**

# Optimisation des performances

## Accélérez les chargements

- ▶ Mettre les scripts en fin de page, et si possible ajouter un script pour charger vos fichiers js après le chargement de la page.

```
<script>  
window.onload = downScripts;  
  
function downScripts() {  
    var element =  
        document.createElement("script");  
    element.src = "myScript.js";  
  
    document.body.appendChild(element);  
}  
</script>
```

# Limites

## Limites

- ▶ Pour le côté pure client impossible de faire le lien avec des bases de données
- ▶ Pas possible de lire ou écrire sur le disque dur (hors cookies)
- ▶ Pas facile à debugger

## Avantages

- ▶ Facile et rapide à prendre en main
- ▶ Langage universel
- ▶ Indispensable pour tous sites web

# TypeScript

- ▶ TypeScript est un sur-ensemble de JavaScript.
- ▶ Il a été développé par Microsoft et intégré dans Visual Studio Ce qui facilite son déploiement et également la gestion des erreurs.
- ▶ Tout code JavaScript est valide et fonctionnel en TypeScript.
- ▶ TypeScript apporte à JavaScript une programmation orientée objet classique.
- ▶ A tester : <http://www.typescriptlang.org/play/>

# TypeScript

- ▶ Les ajouts :
  - ▶ Classe
  - ▶ Interface
  - ▶ L'encapsulation
  - ▶ Typage statique

- ▶ var s = 'Hello';
- ▶ var s: string = 'Hello';
- ▶ s = 123; // Cannot convert 'number' to 'string'.
- ▶ var s: any = 'Hello';
- ▶ s = 123; // ok

# TypeScript

```
class Greeter {  
    greeting: string;  
    constructor(message: string) {  
        this.greeting = message;  
    }  
    greet() {  
        return "Hello, " + this.greeting;  
    }  
}  
let greeter = new Greeter("world");  
let button = document.createElement('button');  
button.textContent = "Say Hello";  
button.onclick = function() {  
    alert(greeter.greet());  
}  
document.body.appendChild(button);
```

```
var Greeter = /** @class */ (function () {  
    function Greeter(message) {  
        this.greeting = message;  
    }  
    Greeter.prototype.greet = function () {  
        return "Hello, " + this.greeting;  
    };  
    return Greeter;  
}());  
var greeter = new Greeter("world");  
var button = document.createElement('button');  
button.textContent = "Say Hello";  
button.onclick = function () {  
    alert(greeter.greet());  
};  
document.body.appendChild(button);
```

# TypeScript

- ▶ Débuter :
  - ▶ <https://www.typescriptlang.org/>
  - ▶ Installer avec node \$ npm install -g typescript
  - ▶ Complier \$ tsc helloworld.ts
  - ▶ IDE :
    - ▶ Visual Studio 2015 et 2017
    - ▶ [Sublime Text](#)
    - ▶ [Atom](#)
    - ▶ [Eclipse](#)
    - ▶ [WebStorm](#)

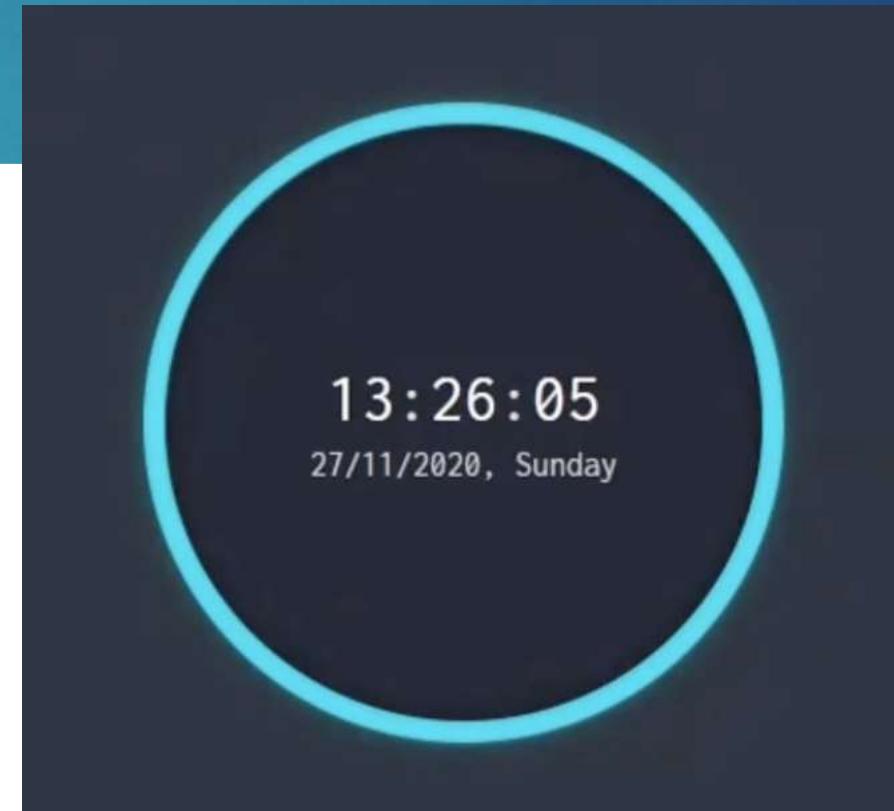
# LIENS

- <http://www.w3schools.com/js/default.asp>
- <https://openclassrooms.com/courses/dynamisez-vos-sites-web-avec-javascript>
- <http://www.commentcamarche.net/contents/577-javascript-introduction-au-langage-javascript>
- <https://developer.mozilla.org/fr/docs/Web/JavaScript>
- <http://maxlab.fr/2014/12/bonnes-pratiques-javascript-pour-lentreprise/>
- [http://www.w3schools.com/js/js\\_best\\_practices.asp](http://www.w3schools.com/js/js_best_practices.asp)

TD : source

<https://www.instagram.com/richwebdeveloper/>

Réalisez une horloge en JS + CSS



```

<div class="container">
  <div class="time">
    <span id="hour">00</span>
    <span class="blink-colon">:</span>
    <span id="minute">00</span>
    <span class="blink-colon">:</span>
    <span id="second">00</span>
  </div>
  <div class="date-container">
    <span id="date">01/01/2014</span>,
    <span id="day">Sunday</span>
  </div>
</div>

```

HTML

```

1 body {
2   font-family: 'Inconsolata', monospace;
3   display: flex; align-items: center;
4   justify-content: center;
5   height: 100vh;
6   background: #2c2f3b;
7 }
8 .container {
9   border: 10px solid rgb(0, 240, 240);
10  width: 300px; height: 300px;
11  display: flex; justify-content: center;
12  align-items: center; flex-direction: column;
13  color:#fff; font-size: 32px;
14  border-radius: 50%;
15  background: rgb(36, 37, 51);
16  box-shadow: 0 0 12px rgba(0,240,240,0.3), 0 0 12px
               rgba(0,0,0,0.4) inset;
17 }
18 .blink-colon {
19   animation: blink 1s infinite;
20 }
21 @keyframes blink {
22   0%, 100% { opacity: 1; }
23   30% { opacity: 0.4; }
24 }
25 .date-container {
26   margin-top: 6px;
27   font-size: 16px;
28   color: rgba(255,255,255,0.8)
29 }

```

CSS-1

CSS-2

```

1 function updateTime() {
2     const hourEL = document.querySelector("#hour");
3     const minuteEL = document.querySelector("#minute");
4     const secondEL = document.querySelector("#second");
5     let d = new Date();
6     let hours = d.getHours();
7     let minutes = d.getMinutes();
8     let seconds = d.getSeconds();
9     if (hours ≥ 0 && hours ≤ 9) hours = "0" + hours;
10    if (minutes ≥ 0 && minutes ≤ 9) minutes = "0" + minutes;
11    if (seconds ≥ 0 && seconds ≤ 9) seconds = "0" + seconds;
12    hourEL.innerHTML = hours;
13    minuteEL.innerHTML = minutes;
14    secondEL.innerHTML = seconds;
15
16    const dateEl = document.querySelector("#date");
17    let date = d.getDate();
18    if (date ≥ 0 && date ≤ 9) date = "0" + date;
19    let month = d.getMonth();
20    if (month ≥ 0 && month ≤ 9) month = "0" + month;
21    let year = d.getFullYear();
22    if (year ≥ 0 && year ≤ 9) year = "0" + year;
23    dateEl.innerHTML = `${date}/${month}/${year}`;

```

JS-1

```

24    const dayEL = document.querySelector("#day");
25    let daynumber = d.getDay();
26    let day = "";
27    switch (daynumber) {
28        case 0: day = "Sunday"; break;
29        case 1: day = "Monday"; break;
30        case 2: day = "Tuesday"; break;
31        case 3: day = "Wednesday"; break;
32        case 4: day = "Thursday"; break;
33        case 5: day = "Friday"; break;
34        case 6: day = "Saturday"; break;
35    }
36    dayEL.innerHTML = day;
37 }
38
39 window.onload = function () {
40     setInterval(updateTime, 1000)
41 };

```

JS-2