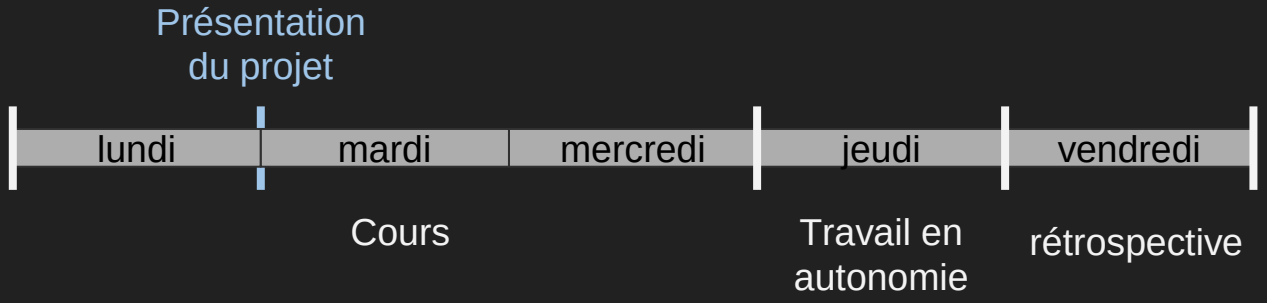
JavaScript Avancé

édition 2022
Coding Factory

Le déroulement de la semaine



- Horaires : 9h-17h
- Pauses : oui
- Notation : oui

Sommaire

- Rappels
  - Variables
  - Tableaux
  - Fonctions
  - BOM et DOM
- Les objets
- Notation des fonctions
- La récursivité
- jQuery
- Découverte de l’AJAX avec jQuery
- Déboguer son code

Variables | Déclaration des variables

var

- La portée est la fonction.
- Si déclarée au niveau global, la variable est ajoutée à l'objet global (window dans un navigateur).
- Déclaration de la variable remontée en début de fonction et initialisée à undefined jusqu'à affectation d'une valeur.
- La variable peut être réaffectée.

```
var maVariable;
```

let

- La portée est le bloc.
- Si déclarée au niveau global, la variable n'est PAS ajoutée à l'objet global.
- Déclaration de la variable remontée en début de bloc, mais variable inaccessible jusqu'à sa "déclaration réelle".
- La variable peut être réaffectée.

```
let maVariable;
```

const

- La portée est le bloc.
- Si déclarée au niveau global, la variable n'est PAS ajoutée à l'objet global.
- Déclaration de la variable remontée en début de bloc, mais variable inaccessible jusqu'à sa "déclaration réelle".
- La variable ne peut pas être réaffectée (mais sa valeur peut être mutée s'il s'agit d'un objet).

```
const maVariable;
```

Variables | Règles de portées

```
function varTest() {
  var x = 31;
  if (true) {
    var x = 71;  // c'est la même variable !
    console.log(x);  // 71
  }
  console.log(x);  // 71
}

function letTest() {
  let x = 31;
  if (true) {
    let x = 71;  // c'est une variable différente
    console.log(x);  // 71
  }
  console.log(x);  // 31
}
```

- Les variables déclarées avec let appartiennent à la portée du bloc dans lequel elles sont définies et indirectement aux portées des blocs de ce bloc.

  D'une certaine façon, let fonctionne comme var. La seule différence est que let fonctionne avec les portées de bloc et var avec les portées des fonctions.

- Au niveau le plus haut (la portée globale), let crée une variable globale alors que var ajoute une propriété à l'objet global.

Variables | Types de variables

- Il n’est pas nécessaire de préciser le type de valeur qu’une variable va pouvoir stocker.

- En JavaScript, il existe 7 types de valeurs différents :

    - String
    - Number
    - Boolean
    - Null
    - Undefined
    - Symbol
    - Object

Tableaux | Créer et accéder

Créer un tableau

```
var fruits = ['Apple', 'Banana'];

console.log(fruits.length);
// 2
```

Accéder au contenu du tableau

```
var first = fruits[0];
// Apple

var last = fruits[fruits.length - 1];
// Banana
```

Tableaux | Boucler

Boucler sur un tableau

```
fruits.forEach(function(item, index, array) {
  console.log(item, index);
});
// Apple 0
// Banana 1
```

```
var newLength = fruits.push('Orange');
// ["Apple", "Banana", "Orange"]
```

Tableaux | Ajouter

Ajouter en fin de tableau

```
var newLength = fruits.push('Orange');
// ["Apple", "Banana", "Orange"]
```

Ajouter en début de tableau

```
var newLength = fruits.unshift('Strawberry') // ajoute au début
// ["Strawberry", "Banana"];
```

Tableaux | Supprimer

Supprimer le dernier élément du tableau

```
var last = fruits.pop(); // supprime Orange (à la fin)
// ["Apple", "Banana"];
```

Supprimer le premier élément du tableau

```
var first = fruits.shift(); // supprime Apple (au début)
// ["Banana"];
```

Tableaux | Rechercher un index et le supprimer

Trouver l'index d'un élément dans le tableau

```
fruits.push('Mango');
// ["Strawberry", "Banana", "Mango"]

var pos = fruits.indexOf('Banana');
// 1
```

Supprimer un élément avec son index

```
var removedItem = fruits.splice(pos, 1); // supprime 1 élément à la position pos

// ["Strawberry", "Mango"]
```

Fonctions | Template et exemple

Les fonctions sont des blocs de code nommés et réutilisables et dont le but est d'effectuer une tâche précise.

Template de déclaration

```
function name(param1, param2, ...) {
    code...

    return (pas obligatoire)
}
```

Exemple

```
function div(a, b) {
  if(b == 0) {
    return 'Divison par 0 impossible';
  } else {
    return a / b;
    alert('Ce message ne s\affiche jamais !');
  }
}
```

BOM et DOM | BOM

Le BOM : Browser Object Model

Le BOM correspond aux données et fonctions rattachées au navigateur. On retrouvera par exemple :

- L’historique
- Les fonctions alert, prompt et confirm
- Le localStorage
- Le sessionStorage
- Les événements dans la fenêtre (scroll, ouvrir une fenêtre ou un onglet, AddEeventListener, …)
- Afficher des informations dans la console
- Le DOM

BOM et DOM | BOM

Le DOM : Document Object Model

Quand un navigateur reçoit une page HTML, qui est reçue sous forme de simple chaîne de caractère, il la décortique et construit ce que l'on appelle le DOM (pour Document Object Model). Et c'est le DOM qui sert de référence pour créer le rendu de votre page, via le moteur de rendu qui va se charger de "dessiner" votre page dans le navigateur.

- Le DOM est une représentation structurée du document sous forme « d'arbre » crée automatiquement par le navigateur.
- Chaque branche de cet arbre se termine par ce qu'on appelle un nœud qui va contenir des objets. On va finalement pouvoir utiliser ces objets, leurs propriétés et leurs méthodes en JavaScript.

BOM et DOM | DOM

- Un modèle objet est créé au chargement de la page. Celui-ci correspond à une autre représentation de la page sous forme d'arborescence contenant des objets qui sont de type Node (nœuds).

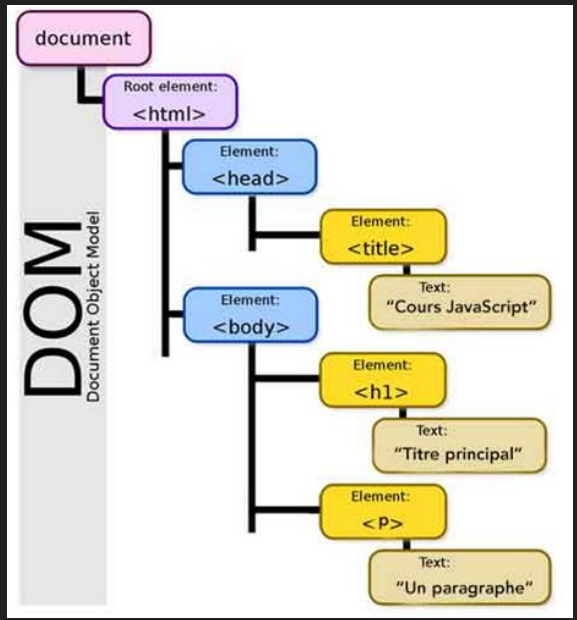
- Les navigateurs utilisent eux-mêmes cette arborescence qui va s'avérer très pratique à manipuler pour eux et notamment pour appliquer les styles aux bons éléments. Nous allons également pouvoir utiliser ce modèle objet en utilisant un langage de script comme le JavaScript.

```
<!DOCTYPE html>
<html>

  <head>
    <title>Cours de JavaScript</title>
    <meta charset="utf-8">
  </head>

  <body>
    <h1>Titre</h1>
    <p>Paragraphe</p>
  </body>

</html>
```



BOM et DOM | Les interfaces composants le DOM

Sélectionner des éléments

BOM et DOM | Principales API du DOM

Ce qui suit est une brève liste des API disponibles :

```
2
3   /* Séléctionner un élément */
4
5   document.getElementById( id );
6   document.getElementsByTagName( nomDuTag );
7   document.getElementsByClassName( nomDeClasse );
8   document.querySelector( idOuClasse );
9   document.querySelectorAll( idOuClasse );
10
11  /* Créer un élément */
12
13  document.createElement( nomElement ) // p, div, a, span, ...
14
15  /* Ajouter un élement à un autre élément */
16
17  let parentNode  = document.getElementById( 'monId' );
18  let element     = document.createElement( 'p' );
19
20  parentNode.appendChild( element );
```

```
22  /* Modifier ou accéder à un attribut */
23
24  element.setAttribute( nomAttribut, valeurAttribut );
25  element.getAttribute( nomAttribut );
26
27  /* Ajouter un événement */
28
29  element.addEventListener( 'click', function( event ) {
30    // Contenu de ma fonction
31  });
32
33  /* Modifier le contenu d'un éléménet */
34
35  element.innerHTML = 'mon nouveau contenu';
36
```

Les objets | Attributs, méthodes & définition

- **Attributs**

  C'est une variable interne à notre classe que l'on peut définir par défaut et faire évoluer au fur et à mesure de l'exécution de notre code.

- **Méthodes**

  Une méthode est une fonction appartenant à notre classe. Celle-ci va pouvoir interagir avec nos attributs de classe.

Le mot clé « **this** » fait référence à l'objet en cours. A chaque fois que l'on instancie un objet, son contenu est unique. « **this** » permet de spécifier que l'on parle du contenu de notre objet.

```
/* Définition de notre classe */

class Personnage {

  variableEnPlus = "test";

  /* Constructeur de notre classe */
  constructor( nom, vie) {
    this.nom = nom;
    this.vie = vie;
  }

  /* Méthode de notre fonction */
  afficherInformations() {
    console.log( 'Le personnage ' + this.nom + ' à une vie de ' + this.vie + ' %' );
    console.log( 'Voici le contenu de la variable en plus : ' + this.variableEnPlus );
  }

}

/* Créer notre personnage, on créé une instance */
var perso = new Personnage( "Jojo", 90 );

/* Afficher les informations du personnage */
perso.afficherInformations();
```

Les objets | Modifier les attributs

- Les **attributs** peuvent évoluer dans le temps s'ils ne sont pas statiques.

  L'objectif est de faire évoluer notre objet dans le temps et au fil de l'exécution de notre programme.

  Dans l'exemple précédent, notre personnage peut évoluer (changer de nom, perdre ou gagner de la vie, etc…)

- C'est avec une **méthode** (fonction) que l'on peut modifier nos attributs

```
/* Définition de notre classe */

class Personnage {

  variableEnPlus = "test";

  /* Constructeur de notre classe */
  constructor( nom, vie) {
    this.nom = nom;
    this.vie = vie;
  }

  /* Méthode de notre fonction */
  afficherInformations() {
    console.log( 'Le personnage ' + this.nom + ' à une vie de ' + this.vie + ' %' );
    console.log( 'Voici le contenu de la variable en plus : ' + this.variableEnPlus );
  }

  /* Méthode pour changer de nom */
  changerDeNom( nouveauNom ) {
    console.log( this.nom + ' s\'appelle désormais ' + nouveauNom );

    // Affectation du nouveau nom
    this.nom = nouveauNom;
  }
}

/* Créer notre personnage, on créé une instance */
var perso = new Personnage( "Jojo", 90 );

/* Afficher les informations du personnage */
perso.afficherInformations();

/* On change le nom de Jojo */
perso.changerDeNom( 'Bob' );
```

Les objets | Méthodes statiques

- Il existe un type spécial de méthode pouvant être ajoutée à une classe : la méthode statique.

  Elle est différente des méthodes d'instance parce qu'**elle n'est pas liée à une instance particulière d'une classe, mais à la classe elle-même.**

  Utilisez-la pour créer des méthodes utilitaires (helper en anglais) où vous n'aurez pas besoin d'une instance d'une classe pour les utiliser. Vous pourrez vous en servir comme boîte à outils de fonctions.

```
2
3  /* Définition de notre classe */
4
5  class Personnage {
6
7    /** Methode statique **/
8    static messageDeBienvenu() {
9      console.log( 'Cette méthode statique est utilisée pour afficher un message' );
10   }
11
12
13 }
14
15 /* Afficher un message de bienvenu */
16 Personnage.messageDeBienvenu();
17
```

Les objets | Intéraction entre objet

- Prenons l'exemple de nos personnages. Si nous souhaitons que le Personnage 1 vienne guérir le Personnage 2 pour qu'il ait toute sa vie, il nous faut faire communiquer nos objets ensemble.

  Comment faire cela ?

  C'est très simple ! Avec une méthode et un objet en paramètre.

- On accède aux attributs d'un objet en paramètre d'une méthode

```
5  class Personnage {
6
7    variableEnPlus = "test";
8
9    /* Constructeur de notre classe */
10   constructor( nom, vie, type ) {
11     this.nom  = nom;
12     this.vie  = vie;
13     this.type = type;
14   }
15
16   /* La méthode guerir prend un objet personnage en paramètre */
17   guerir( personnage ) {
18     if( this.type == 'Médecin' ) {
19       personnage.vie = 100;
20
21       console.log( this.nom + ' vient de guérir ' + personnage.nom );
22     }else {
23       console.log( this.nom + ' n\'est pas médecin !' );
24     }
25   }
26 }
27
28 /* Afficher un message de bienvenu */
29 Personnage.messageDeBienvenu();
30
31 /* Créer notre personnage, on créé une instance */
32 var perso   = new Personnage( "Jojo", 4, 'Soldat' );
33 var perso2  = new Personnage( "Bob", 100, 'Médecin' );
34
35 // Impossible
36 perso.guerir( perso2 );
37
38 // Le médecin va guérir le soldat
39 perso2.guerir( perso );
```

Notation des fonctions

Définir une fonction JavaScript, plusieurs notations

```
28
29  /* Notation 1 : Simple */
30  function afficher( parametre ) {
31    console.log( parametre );
32  }
33  afficher( "Bonsoir !" );
34
35  /* Notation 2 : fonctions fléchées */
36  let somme = (a, b) => (a + b);
37  console.log( somme( 56, 68 ) );
38
39  /* Notation 3 */
40  let carre = function( parametre ) {
41    return parametre * parametre;
42  }
43  console.log( carre( 10, 10 ) );
44
```

La récursivité

- La récursivité est un grand concept de la programmation : il s'agit d'une méthode permettant d'obtenir une solution à un problème en utilisant des instances plus petites de ce même problème. En clair, une fonction est donc dite récursive si elle s'appelle elle-même.

- Une fonction récursive est une boucle . Tout comme une boucle **for** ou une boucle **while**. Tout ce que nous essayons de faire est de répéter du code jusqu'à ce qu'une condition soit remplie

- Dans le cas présent, nous comptons simplement jusqu'à 100.

- La récursivité est souvent utile dans la recherche d'éléments dans un tableau ou pour trier son contenu. Bien évidemment, la récursivité peut être utilisé pour bien d'autres cas.

```
1  /* Compter jusqu'à 100 */
2
3  function compterCent( valeur ) {
4
5    // Afficher la valeur
6    console.log( valeur );
7
8    if( valeur >= 100 ) return valeur;
9
10   // Incrémenter
11   valeur++;
12
13   // Appel récursif
14   compterCent( valeur );
15 }
16
17 compterCent( 0 );
```

jQuery et le DOM | Introduction

Introduction

jQuery est une bibliothèque JavaScript, c'est à dire un ensemble de fonctions, permettant de manipuler très facilement et de façon très concise l'arbre DOM. Cette librairie vient par ailleurs avec de nombreux plugins facilitant le développement.
Également, jQuery gomme une partie des incompatibilités entre les navigateurs internet, en particulier avec AJAX, rendant le code plus portable.
Et on peut utiliser les fonctions jQuery dans n'importe quel programme JavaScript

Importer JavaScript

Il existe 2 solutions :

- Télécharger le fichier .js minifié sur le site de jQuery et le placer dans le dossier de notre site internet. A partir de là, nous chargerons le fichier depuis une balise JavaScript.

- Récupérer l'URL du fichier depuis le site de jQuery et placer cette url dans une balise JavaScript, de la même manière que précédemment.

jQuery et le DOM | Les sélecteurs

Il existe plusieurs manières de faire une sélection :

- **La sélection des nœuds :**
  - Par nom de d'élément (balise)
  - Par identifiant
  - Par classe
- **En tenant compte de la hiérarchie :**
  - Tous les descendants
  - Descendants directs
  - Composition sur plusieurs niveaux
- **En tenant compte des attributs :**
  - Test de l'existence
  - Test de la valeur d'un attribut

```
/*
* 1- Sélection des noeuds
*/

$( "h1" ); // Par nom d'élément
$( "#mon-id"); // Par id
$( ".ma-classe" ); // Par classe

/*
* 2- En tenant compte de la hiérarchie
*/

$( "h1 span" ); // Tous les descendants
$( "h1 > span" ); // Descendant direct
$( "h1 > p > span" ); // Composition

/*
* 3 - En tenant compte des attributs
*/

$( 'input[type]' );
$( 'input[type="text"]' );
```

jQuery et le DOM | La fonction find()

Il est tout à fait possible d'effectuer une recherche sur la racine d'un arbre DOM encapsulé dans un objet jQuery

```
/*
* Recherche
*/

/*
* Avec ceci, l'ensemble des balise span contenu dans la balise ayant
* l'id "mon-id"
*/
$( '#mon-id' ).find( 'span' );
```

jQuery et le DOM | Filtre

jQuery définit également des versions « raccourcies » de certaines expressions de sélection souvent utilisées, notamment en ce qui concerne les formulaires.

Dans l'expression d'un sélecteur :

```
$( "selecteur:filtre" );
```

Avec la fonction « filter » :

```
$("selecteur").filter(":filtre");
```

Ne garde de la sélection que les éléments qui passent le filtre.

Avec la fonction « not » :

```
$("selecteur").not(":filtre");
```

Ne garde de la sélection que les éléments qui ne passent pas le filtre.

Exemples de filtres :

- Type d'élément :
  :input, :checkbox, :radio, :password
- État d'élément :
  :focus, :checked, :selected
- Ou encore :
  :even, :odd, :empty, :contains(texte)

Les filtres également peuvent se composer :

```
$("input").filter(":checkbox").not(":checked");
```

jQuery et le DOM | Opération sur les collections

Il est possible d'obtenir la taille de la collection :

```
$("td").length;
```

Et également de sélectionner des éléments de la collection :

```
$("td").length;

$("td").first() // Premier élément

$("td").last()  // Dernier élément

$("td").eq(i)   // Ième élément
```

jQuery et le DOM | Manipulation du DOM

Avec jQuery, on peut faire toutes les opérations usuelles sur le DOM de façon plus concise.

A partir d'un élément de l'arbre, on peut :

```
/* Accéder aux attributs */
.attr("nomAttr");

/* Accéder aux parents, aux ancêtres */
.parent();
.parents();

/* Accéder aux enfants */
.children();

/* Accéder aux frères */
.siblings();
.next();
.prev();
```

```
/* Accéder au texte de l'élément */
.text();

/* Accéder à l'ensemble du sous-arbre sous forme de texte avec balises */
.html();

/* Accéder à la valeur d'un nœud <input>, <select> ou <textarea> */
.val();
```

jQuery et le DOM | Modification de l'arbre

A partir d'un élément de l'arbre :

```
/* Insertion/modification d'attribut */
.attr( "nomAttr", "valeur" );
```

Insertion :

```
/* D'enfants */
.append( element );
.prepend( element );

/* De frères */
.after( element );
.before( element );

/* De parents */
.wrap( element );

/* De texte */
.text( element );

/* D'un sous-arbre entier */
.html( element );

/* Remplacement de l'élément par un autre */
.replaceWith( element );

/* Suppression de l'élément */
.remove();
.detach();
```

jQuery et le DOM | Style CSS

Propriétés CSS individuellement

```
19
20  /* Lecture */
21  .css( "nomPropriete" );
22
23  /* Modification */
24  .css( "nomPropriete", "valeur" );
25
26  /* Modification de plusieurs propriétés */
27  .css( {"prop1": "val1", "prop2": "val2", ...} );
```

Ajout/suppression de classes CSS

Il faut que ces classes soient déjà existantes.
Il est impossible de manipuler le contenu de style .css avec jQuery.

```
20  .addClass( "nomClasse" );
21  .removeClass( "nomClasse" );
```

jQuery et le DOM | Les événements

L’événement « ready »

En javascript standard, l'événement load est émis lorsque toute la page est prête.
jQuery propose une alternative : l'événement ready. Celui est émis lorsque tous les éléments du DOM sont définis, mais que le contenu n'est pas encore là. Donc c'est un événement qui a lieu avant l'événement load.

Attention, avec jQuery :

- **load** est associé à la fenêtre (object window)
- **ready** est associé au document (object document)

Associer un événement

Pour associer une action à un objet (ou une collection d'objets) jQuery en fonction d'un événement, on utilise :

```
$( '#mon-id' ).nomEvenement( function( event ) {
  // Contenu de la fonction
});

$( document ou element ).on( 'nomEvenement', 'cible (facultatif)', function( event ) {
  // Contenu de la fonction
});

/* Exemple */

$( '#mon-id' ).on( 'click', '.ma-classe', function( event ) {
  alert( 'click !' );
});
```

Liste des événements

Vous pouvez retrouver la liste de tous les événements en suivant ce lien :

https://api.jquery.com/category/events/

Ajax | jQuery

Le terme AJAX est l'abréviation de "Asynchronous JavaScript and XML". L'AJAX n'est pas un langage de programmation mais correspond plutôt à un ensemble de techniques utilisant des technologies diverses pour communiquer avec le serveur (notamment envoyer et récupérer des données) de façon asynchrone, c'est-à-dire sans avoir à recharger la page.

Avant l'utilisation de code asynchrone et l'AJAX, la moindre action de l'utilisateur (envoi ou demande de données) résultait par le chargement d'une nouvelle page envoyée par le serveur. Ce processus était inefficace, lent, et peu agréable pour l'utilisateur.

Ajax | jQuery

- Méthode d’envoi

  POST ou GET

- Format de données

  - text : permet de transporter du texte
  - html : permet de transporter du contenu HTML
  - json : permet de transporter un élément au format JSON
  - script : permet de transporter du script JS pour l’ajouter à la page
  - jsonp : permet de transporter un élément au format JSON depuis un autre domaine
  - xml : permet de transporter du contenu XML

Ajax | jQuery

Nous allons pouvoir passer plusieurs options à $.ajax() afin de fournir des instructions à notre requête. Les options les plus couramment utilisées sont les suivantes :

url : URL de la requête. **Obligatoire**.

method : (Par défaut : GET)

dataType : Type de données attendu en réponse.

data : Données à envoyer au serveur.

async (Par défaut : true) : Requête asynchrone (true) ou synchrone (false) ;

cache (Pdéfaut : true sauf si le type de données défini est script ou jsonp) : Utiliser une réponse en cache si disponible (true) ou pas (false)

beforeSend : Permet de définir des en-têtes personnalisées

name : Nom d'utilisateur à utiliser dans le cas où une demande d'identification HTTP est fait

password : Mot de passe à utiliser dans le cas où une demande d'identification HTTP est faite

timeout : Délai d'attente en millisecondes avant de considérer la demande comme un échec.

Ajax | jQuery

```
$( document ).ready( function() {

  $.ajax({
    // L'url de la requête
    url: 'monUrl/vers/mon/fichier/php/ou/autre',
    // La methode d'envoie
    method: 'POST',
    // Le format de réponse attendu
    dataType: 'json',
  })
  // Ce code sera exécuté en cas de succès
  .done(function( response) ) {
    let data = JSON.stringify(response);

    // Tâches à exécuter
  })
  // Ce code sera exécuté en cas d'échec
  .fail(function(error) {
    alert( 'La requête s\'est terminée en échec. Infos : ' + JSON.stringify(error) );
  })
  // Ce code sera exécuté que la requête soit un succès ou un échec
  .always(function() {
    alert( 'Requête exécutée' );
  });

});
```

Déboguer son code

Il y a plusieurs éléments sur lesquels être vigilant lorsque vous développez :

- **Ouvrez la console et assurez vous qu'aucune erreur n'est remontée**.
  Bien souvent, il y a une erreur associée à un comportement non désiré de notre programme.

- **Utiliser autant que possible console.log** pour vous assurer de la cohérence du déroulement de votre code à des intervalles clés.

  Si le résultat final ne vous convient pas, assurez vous que le résultat est bon quelques lignes plus haut.

  console.log est votre allié pour déboguer !

- **Vérifier la logique de votre code.**

  Et oui ! Bien souvent, l'erreur est purement logique. Un calcul avec une chaine de caractère au lieu d'un entier, une boucle sur un tableau vide, …
  Tout dépend de votre code ! Relisez et parlez en avec vos collaborateurs.

- **Relisez votre code, ligne après ligne, à une personne tiers ou à haute voix.**
  Reprendre à voix haute et par étape va vous aider à identifier le problème.

```
28
29  /* Afficher un titre puis le contenu de la variable */
30  console.log( 'titre', maVariable );
31
32  /* Afficher le contenu simple de la variable */
33  console.log( maVariable );
```