

Intégration Web

MMI 1 – TP#4 S2



Danielo **JEAN-LOUIS**
Michele **LINARDI**

javascript

- Naissance en 1995
 - Apparemment écrit en une semaine
- Langage de programmation côté client
 - Et serveur depuis 2008

javascript

- Principale utilisation : interaction sur une page web. Exemples :
 - Clic sur un bouton
 - Focus clavier dans un champ de formulaire
- Langage dit événementiel

- Peut être écrit directement dans un fichier html
 - balise `<script> /* code */ </script>`
- Ou chargement d'un fichier distant
 - balise `<script src="chemin-fichier.js"></script>`
 - **Préférer cette méthode pour des questions de lisibilité**
- Fichier lu de haut en bas par le navigateur

Point technique : Où charger ses scripts ?

- Deux possibilités :
 - Dans la balise `<head>` avec l'attribut "defer"
 - Méthode moderne → A privilégier
 - Avant la fermeture de la balise `<body>`

Point technique : Où charger ses scripts ?

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!-- Code HTML -->
  <script src="mon-script.js"></script>
</body>
</html>
```

Point technique : Où charger ses scripts ?

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>

  <script src="mon-script.js" defer</script>
</head>
<body>
  <!-- Code HTML -->
</body>
</html>
```


Point technique : Où charger ses scripts ?

- Chargement dans la balise <head>
 - Utilisation **impérative** de l'attribut "defer" dans la balise <script>
- Chargement avant la fermeture de <body>
 - Utilisation non-impérative

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/HTML/Element/script#attr-defer>

- Extension de fichier ".js"
- langage faiblement typé :
 - Un entier peut devenir une chaîne de caractères !

Langage extrêmement permissif. Soyez rigoureux dans votre code !

Variables

- Non typée
- Permettent de contenir des valeurs
- Préférer un nommage clair et descriptif

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables#la_diff%C3%A9rence_entre_var_et_let

Variables – mot-clé

- constante : mot-clé "const"
 - **Impossibilité** de réaffectation
- variable : mot-clé "let"
 - **Possibilité** de réaffectation

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables#la_diff%C3%A9rence_entre_var_et_let

Variables

```
const jeSuisUneConstante = "je ne changerai  
let jeSuisUneVariable = "Je peux changer"  
  
/* A ne pas utiliser */  
var jeSuisUneVariableAncienneVariation = "";
```

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables#la_diff%C3%A9rence_entre_var_et_let

Point technique : Mot-clé "var"

- Ancienne façon de déclarer une variable
 - Crée plein d'effet de bord (hissage)

N'utilisez pas le mot-clé "var" pour déclarer vos variables

Source(s) :

- <https://developer.mozilla.org/fr/docs/Glossary/Hoisting>

Variables – Types possibles

- Nombre (décimal ou entier)
- Chaîne de caractères
- Booléen
- Tableau
 - Permet de contenir plusieurs valeurs / variables
- Objet
- nul (null) / indéfini (undefined)
- Fonction

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables#la_diff%C3%A9rence_entre_var_et_let

Fonctions

- Permettent de réutiliser le code
- Permettent de mieux séparer le code
 - Meilleure lisibilité
- Contiennent un ensemble d'instructions

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Fonctions>

Fonctions

```
// Déclaration de fonction
function maFonction(parametre) {
    let maVariable = parametre;
    // mes instructions
    return maVariable;
}

// Appel de la fonction
maFonction("BUT MMI");
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions>

Fonctions – mot-clé "return"

- Désigne la réponse d'une fonction
 - Possible uniquement dans une fonction
- Présence multiple de return possible dans une fonction
 - Une fonction ne peut retourner qu'un seul élément à la fois

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions>

Une fonction n'est pas obligée de retourner quelque chose

Fonctions – Paramètres de fonction

- Appelés également "arguments"
- Existents uniquement dans la fonction
- Valeurs définies lors de l'appel de la fonction
- Séparés par une virgule

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions>

Fonctions – Paramètres de fonction

```
function soustraction(arg1, arg2) {  
    // arg1 et arg2 existent ici  
    return arg1 - arg2;  
}  
  
soustraction(8, 16);  
// arg1 et arg2 n'existent pas ici
```

On définit deux arguments à notre fonction "soustraction". Ces deux paramètres ne sont accessibles que dans la fonction "soustraction"

On appelle notre fonction les valeurs 8 et 16. L'ordre des paramètres et le même que celui des valeurs passées :

- arg1 = 8
- arg2 = 16

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions>

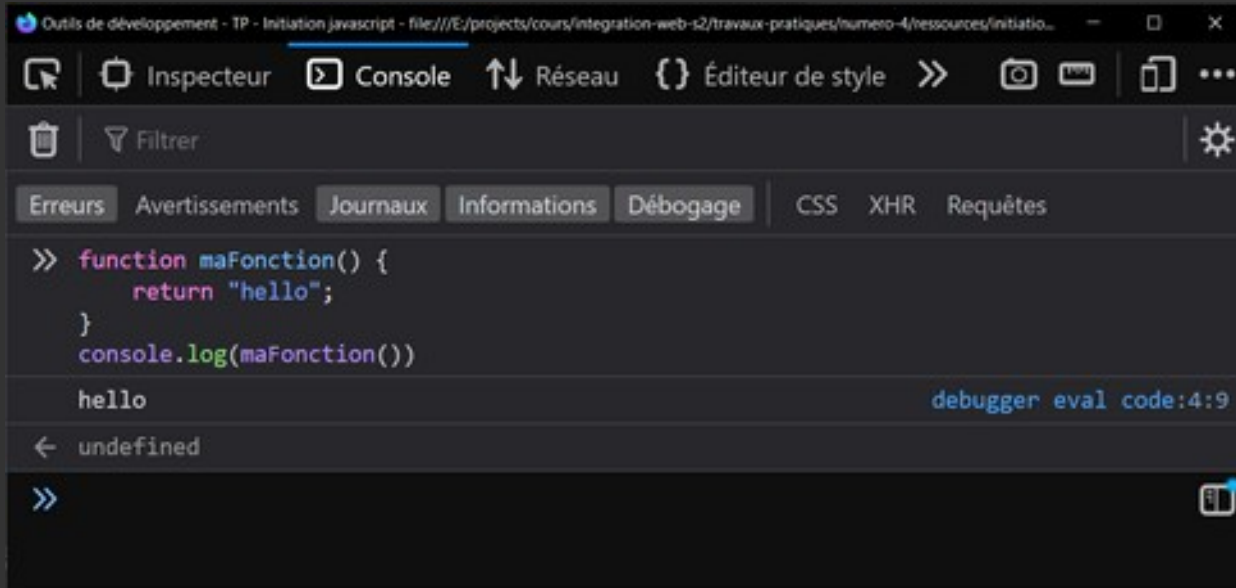
Débugger son code

- Utilisation de la fonction : `console.log()`
 - Permet de debugger son code
- Utilisation de la console du navigateur (touche F12)
 - Onglet "Console"

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Console/log>

Débugger son code




Il est possible écrire du code javascript directement dans la console dans l'onglet "Console"

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Console/log>

Fonctions



```
const hello = "world";  
  
// Affichera "world" dans la console du navigateur  
console.log(hello)
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Console/log>

Point technique : `console.log()` et `alert()`

- Les deux permettent de debugger le code
- `alert()` : visible pour tous utilisateurs
- `console.log()` : visible avec la console du navigateur

Point technique : `console.log()` et `alert()`

- `alert()` est une **action bloquante** :
 - On ne peut rien faire d'autre tant que `alert()` n'a pas disparu
- **Préférer `console.log()` à `alert()`**

Ne laissez pas un alert() de débuggage en production. Jamais !

Pratiquons ! - Initiation javascript (Partie 1/2)

Pré-requis :

- Avoir la ressource `ressources/initiation-javascript`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fintegration-web-s2%2Ftravaux-pratiques%2Fnumero-4%2Fressources%2Finitiation-javascript>

Les fonctions en résumé



```
function maFonction(){  
  // instructions  
  return "Hello"  
}
```

Définition de la fonction



```
const maVar = maFonction()
```

Appel de la fonction
(le résultat est stocké dans une variable)

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions>

Bonne pratique : Les fonctions pures

- Deux caractéristiques :
 - Retourne toujours la même valeur pour les mêmes valeurs
 - Ne modifie pas des variables à l'extérieur de sa portée
- Ne provoquent pas d'effets de bord

Source(s) :

- https://fr.wikipedia.org/wiki/Fonction_pure

Bonne pratique : Les fonctions pures

```
function fonctionPure(a, b) {  
  return a + b;  
}  
  
const somme = fonctionPure(4, 5)
```



Fonction pure

Ne modifie pas des variables à l'extérieur de sa portée

```
let somme = 0;  
function fonctionImpure(a, b) {  
  somme = a + b  
}  
  
fonctionImpure(4, 5)
```



Fonction impure

Modifie une variable à l'extérieur de sa portée. Crée des effets de bord

Source(s) :

- https://fr.wikipedia.org/wiki/Fonction_pure


Une fonction ne peut pas toujours être pure

Point technique : Le point-virgule (;)

- Désigne la fin d'une instruction
 - **Caractère facultatif**
 - Le retour à la ligne suffit

Conditions (if, else if, else)

- Permet de tester une condition et exécute les instructions si la condition est vraie



```
if(maVariable === "MMI") {  
    console.log("Bonjour MMI");  
} else {  
    console.log('Bonjour autre');  
}
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/if...else>
- https://fr.wikipedia.org/wiki/Instruction_conditionnelle_%28programmation%29

Point technique : Le triple égal (===)

- Permet de tester la valeur ET le type
 - `1 == "1" = vrai`
 - `1 === "1" = faux`
 - `1 === Number("1") = vrai`
- A préférer pour éviter de mauvaises surprises

Point technique : Le triple égal (===)

- La fonction `Number()` permet de forcer le type d'une variable en nombre (entier ou décimal)
- La fonction `String()` fait la même mais pour les chaînes de caractères

Conditions (if, else if, else)

- "else if" permet d'ajouter des conditions supplémentaires
- Chaque block (if, else if, else) est exclusif, si on entre dans le if, on ne rentrera pas dans le else, etc.

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/if...else>
- https://fr.wikipedia.org/wiki/Instruction_conditionnelle_%28programmation%29

Conditions (if, else if, else)

```
if(maVariable === "MMI") {  
    console.log("Bonjour MMI");  
} else if(maVariable === "TC") {  
    console.log("Bonjour TC");  
} else if(maVariable === "GE2I") {  
    console.log("Bonjour GE2I");  
} else {  
    console.log('Bonjour autre');  
}
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/if...else>
- https://fr.wikipedia.org/wiki/Instruction_conditionnelle_%28programmation%29

Conditions (if, else if, else) multiples

- && : Et logique. Toutes les conditions doivent être remplies
- || : Ou logique. Une des conditions doit être remplie
- !== : Différent de
- > Strictement supérieur à
 - >= Supérieur ou égal à
- < Strictement inférieur à
 - <= Inférieur ou égal à

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/if...else>
- https://fr.wikipedia.org/wiki/Instruction_conditionnelle_%28programmation%29

Conditions (if, else if, else) multiples

```
// Si les deux conditions sont remplies alors
// on entre dans la condition
if(maVariable === "MMI" && monAge > 18) {
    console.log("Bonjour MMI");
}
/* ... */
```

Et logique (&&) : on affichera "Bonjour MMI" si les deux conditions sont remplies

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Logical_AND

Conditions (if, else if, else) multiples

```
// Si une des deux conditions est remplie alors  
// on entre dans la condition  
if(maVariable === "MMI" || monAge > 18) {  
    console.log("Bonjour MMI");  
}  
/* ... */
```

Ou logique (||) : on affichera "Bonjour MMI" si une des deux conditions est remplie

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Logical_OR

Conditions (if, else if, else) multiples



```
// On peut combiner plusieurs conditions
const aLeBac = true;
if(
  (maVariable === "MMI" || monAge > 18) &&
  aLeBac
) {
  console.log("Bonjour MMI");
}
/* ... */
```

Dans quel cas, on affichera "Bonjour MMI" ?

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Logical_OR

Conditions (if, else if, else) multiples

- Mettre en parenthèses les conditions qui vont ensemble
- Possibilité d'imbriquer des structures if/else if / else dans d'autres structures

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/if...else>
- https://fr.wikipedia.org/wiki/Instruction_conditionnelle_%28programmation%29

Pratiquons ! - Initiation javascript (Partie 3)

Pré-requis :

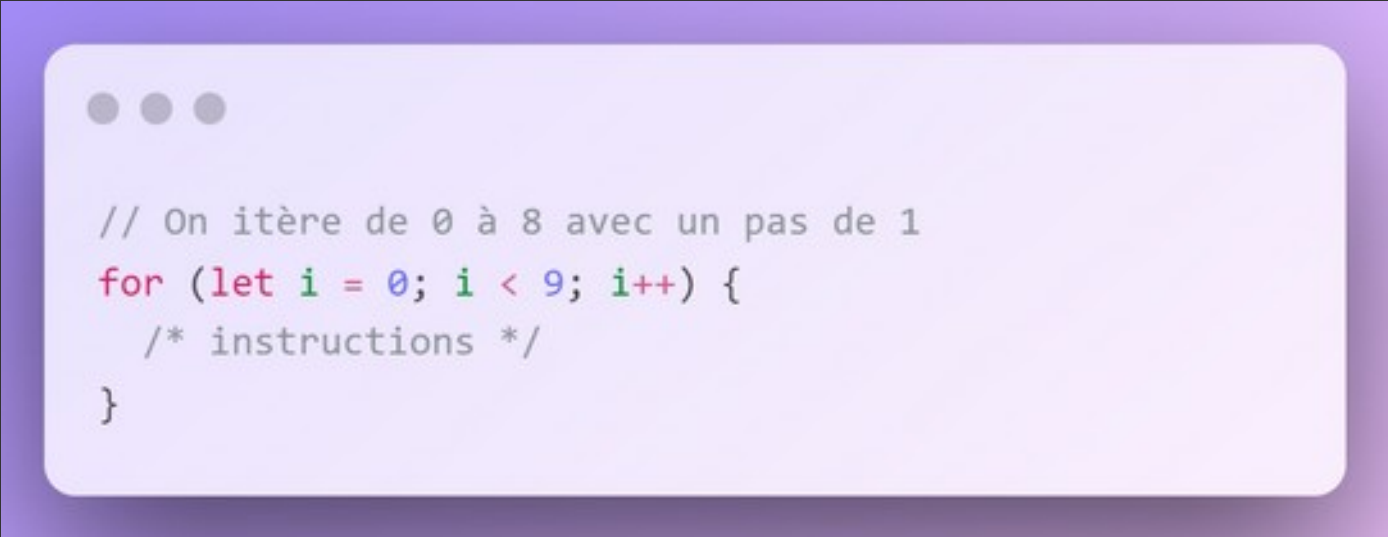
- Avoir la ressource `ressources/initiation-javascript`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fintegration-web-s2%2Ftravaux-pratiques%2Fnumero-4%2Fressources%2Finitiation-javascript>

Boucle for

- Permet de répéter une action un nombre n de fois



```
// On itère de 0 à 8 avec un pas de 1
for (let i = 0; i < 9; i++) {
  /* instructions */
}
```

Source(s) :

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

Boucle for

- Une boucle est bloquante
 - Tant qu'elle n'est pas finie le code après elle ne s'exécutera pas
- La variable itératrice n'existe que dans la boucle
 - La variable "i" n'existe pas à l'extérieur de la boucle for

Source(s) :

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

Boucle for

- Il existe également les boucles while et do...while

Source(s) :

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

Évènements

- Permettent d'interagir avec la **page courante**
- Multitude d'évènements possibles
 - **Vous n'avez pas à les apprendre par cœur**

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/Events>

Évènements – Liste (non exhaustive)

- clic / toucher → "click"
- focus → "focus" (quand on clique dans un champ)
- perte de focus → "blur" (quand on clique hors d'un champ)
- survol → "mouseover"
 - Quelle précaution **devons**-prendre par rapport à cet évènement ?

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/GlobalEventHandlers>

Évènements – Liste (non exhaustive)

- changement dans un élément de formulaire → "change"
- soumission d'un formulaire → "submit"
- Pression sur une touche de clavier → "keydown"

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/GlobalEventHandlers>

Évènements

- Certains évènements ne sont pas compatibles avec certaines balises
- Un évènement doit être lié à un élément HTML

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/Events>

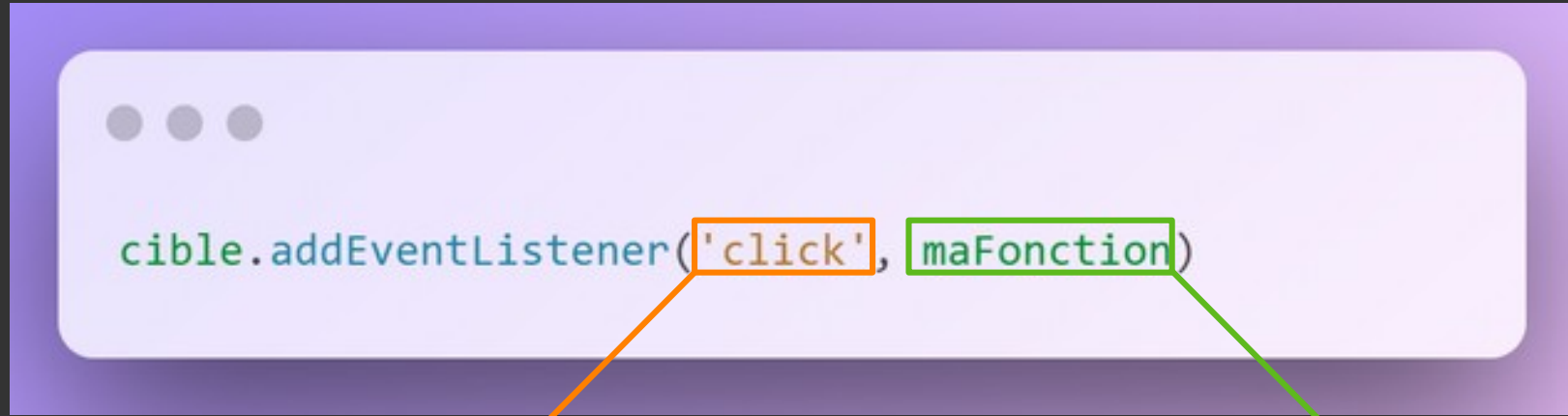
Évènements – Ecouteur d'évènement

- Utilisation de la méthode : "addEventListener"
 - Deux paramètres au minimum :
 - type d'évènement
 - fonction (qu'est-ce qu'il doit se passer lorsque l'évènement est réalisé)
 - **Cette fonction n'a pas besoin de retourner quelque chose**
- Doit être lié à une cible (balise HTML)

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener>

Évènements – Ecouteur d'évènement



Type d'évènement

Nom de la fonction à appeler lors que l'évènement est réalisé (fonction d'évènement)

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener>

Évènements – Fonction d'évènement

- Fonction appelée lorsque l'évènement se produit
- La fonction prend en paramètre l'évènement lui-même :
 - Permet de récupérer l'élément qui a initié l'évènement

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener>

Évènements – Fonction d'évènement

```
function maFonction(evt) {  
    // Contient l'élément qui a déclenché l'évènement  
    // On peut changer son texte (.textContent) par exemple  
    // ou encore son style (.style.nom-propriété-css)  
    console.log(evt.target)  
}  
  
// Retourne le premier <button> trouvé  
const cible = document.querySelector('button')  
cible.addEventListener('click', maFonction)
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener>

Évènements – Cible d'un évènement

- Élément qui va provoquer l'appel d'un évènement
 - Si le bon évènement est déclenché
- Méthodes pour récupérer les éléments :
 - `document.querySelector(sélecteur)`
 - `document.querySelectorAll(sélecteur)`

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector>
- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelectorAll>

Évènements – Cible d'un évènement

- `document.querySelector(sélecteur)`
 - Retourne le **premier** élément trouvé
- `document.querySelectorAll(sélecteur)`
 - Retourne **tous** les éléments trouvés

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector>
- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelectorAll>

Point technique : Le DOM

- Signifie Document Object Model
- Représente toutes les balises de la page sous forme d'objets :
 - document (js) → page entière (html)
 - *Ecrivez "document" dans la console du navigateur (onglet "console") et vous verrez le code entier de la page*

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model/Introduction

Point technique : Le DOM

- Permet d'ajouter / modifier les attributs d'une balise HTML
- Permet d'ajouter / supprimer des balises

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model/Introduction

Point technique : Le DOM



```
// HTML
```

```

```

```
// Javascript
```

```
// On récupère la première balise <img> trouvée
```

```
const element = document.querySelector('img');
```

```
// On change la valeur de l'attribut "src" de la balise <img>
```

```
element.src = 'mon-autre-image.gif';
```

Évènements – Cible d'un évènement

- Les deux méthodes prennent en paramètre un sélecteur CSS
 - Comme ceux utilisés en CSS

```
// Cible le premier élément trouvé ayant la classe CSS "ma-classe"  
// Et le fait appeler la fonction "maFonction" quand on clique dessus  
document.querySelector(".ma-classe").addEventListener('click', maFonction)
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector>
- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelectorAll>

Évènements – querySelectorAll()

- Impossibilité de lier un évènement sur la méthode
 - Nécessite l'utilisation d'une boucle pour lier les évènements un à un

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector>
- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelectorAll>

Évènements – querySelectorAll()

```
const listeElementsMaClasse = document.querySelectorAll(".ma-classe");
for (let i = 0; i < listeElementsMaClasse.length; i++) {
  // On récupère un à un les éléments ayant la classe "ma-classe"
  // on leur assigne l'évènement "click" appelant la classe "maFonction"
  listeElementsMaClasse[i].addEventListener('click', maFonction)
}
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector>
- <https://developer.mozilla.org/fr/docs/Web/API/Document/querySelectorAll>

Point accessibilité : L'évènement click

- Ne doit pas être mis sur la balise `<a>` à la place de la balise `<button>`
 - **Jamais**
- Attention à l'évènement click sur les éléments qui n'ont pas de focus naturel
 - Exemple : La balise `<article>`
 - ça exclut les utilisateurs au clavier

Évènements – Liste (non exhaustive)


- clic → "click"
- focus → "focus"
- perte de focus → "blur"
- survol → "mouseover"
- changement dans un élément de formulaire → "change"
- soumission d'un formulaire → "submit"
- Pression sur une touche de clavier → "keydown"

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/GlobalEventHandlers>

Évènements – Attribut HTML

- Possibilité de lier un évènement via un attribut html

A code editor window with a light purple background and a darker purple border. It features three gray window control buttons (minimize, maximize, close) in the top-left corner. The editor contains HTML and JavaScript code for an inline event handler.

```
<button onclick="maFonction()">Mon bouton</button>
<script>
  function maFonction() {
    alert("hello")
  }
</script>
```

Cette méthode est à éviter

Pratiquons ! - Initiation javascript (Partie 4)

Pré-requis :

- Avoir la ressource `ressources/initiation-javascript`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fintegration-web-s2%2Ftravaux-pratiques%2Fnumero-4%2Fressources%2Finitiation-javascript>

Évènements – Suppression d'un évènement

- Arrêter l'écoute d'un évènement
 - Et d'une fonction associée

```
const cible = document.querySelector('.ma-classe');

// Retire l'appel de la fonction "maFonction"
// au clic sur ".ma-classe"
cible.removeEventListener('click', maFonction);

// Retire l'appel de toutes les fonctions
// au clic sur ".ma-classe"
cible.removeEventListener('click');
```

Data-attribute

- Permettent de créer des attributs personnalisés
- Doivent toujours commencer par "data-"
 - La suite est arbitraire. Ex : data-mmi
- Peuvent avoir une valeur
 - data-mmi="2010-2012"
 - Valeur récupérable en javascript

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/HTML/Howto/Use_data_attributes

Data-attribute

- Doivent être utilisés pour cibler les éléments dans le javascript
 - **Les classes sont faites pour le style**
- Une balise peut avoir un nombre infini de data-attributes uniques
- Plusieurs balises peuvent le même data-attributes

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/HTML/Howto/Use_data_attributes

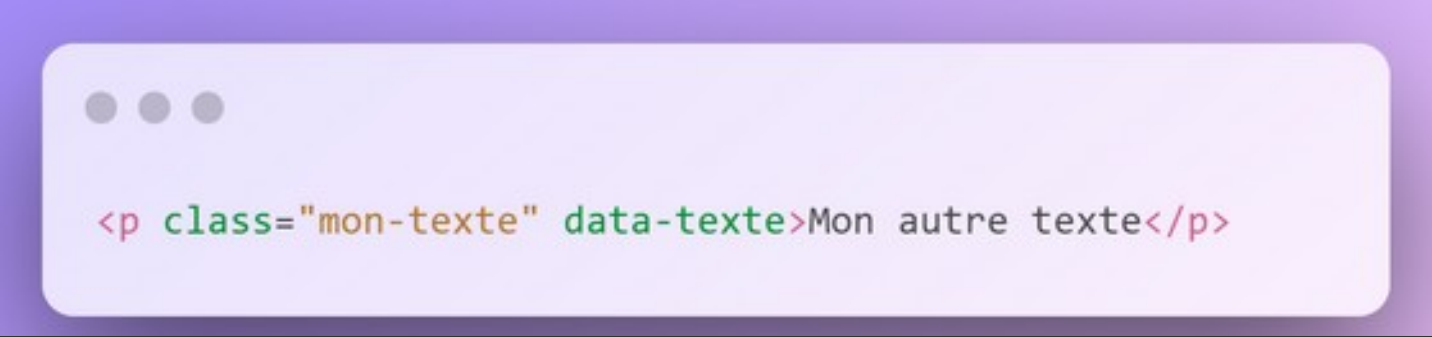
Data-attribute

- Ne peuvent pas avoir d'espaces dans le nom
 - Les espaces peuvent être remplacés par des tirets
 - Ex : "data mmi sar" → "data-mmi-sar"
- La valeur ne peut être qu'entre guillemets
 - Simples (') ou doubles (")

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/HTML/Howto/Use_data_attributes

Data-attribute



...
`<p class="mon-texte" data-texte>Mon autre texte</p>`

Une balise HTML avec un data-attribute



...
`<p class="mon-texte" data-texte="valeur">Mon autre texte</p>`

Une balise HTML avec un data-attribute avec une valeur

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/HTML/Howto/Use_data_attributes

Point technique : Les bons rôles

- data-attribute : ciblage pour le javascript
- classes CSS : ciblage pour le style CSS
- id : pour les ancres et éléments de formulaires

Pratiquons ! - Initiation javascript (Partie 5)

Pré-requis :

- Avoir la ressource `ressources/initiation-javascript`

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fintegration-web-s2%2Ftravaux-pratiques%2Fnumero-4%2Fressources%2Finitiation-javascript>

Questions ?

