

Intégration Web

MMI 2 – TP#1 S3

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

javascript

- Naissance en 1995
- Langage de programmation côté client
 - Et serveur depuis 2008 (via Nodejs)
- Également appelé "js"
- Extension des fichiers ".js"
- Présent sur quasiment tous les sites de nos jours

javascript

- Principale utilisation : Interaction sur les pages web
- Reprend les concepts d'autres langages : variables, fonctions... avec une autre syntaxe
- Langage dit événementiel
- Code lu de haut en bas

Programmation événementielle

- Paradigme de programmation
- Le code réagit en fonction d'évènements :
 - Clic, survol, retour de serveur...

javascript

- Trois façons de charger un fichier js :
 - Dans une balise `<script>`
 - Attribut html : à éviter
 - Fichier externe
 - **Préférer cette méthode pour des questions de lisibilité**
- Les méthodes peuvent être combinées au sein du même fichier html

Chargement des scripts - balise <script>



```
<!-- [...] -->
```

```
<body>
```

```
    <!-- [...] -->
```

```
    <script>/* Mon code javascript */</script>
```

```
</body>
```

```
<!-- [...] -->
```

En mettant des balises <script>, il est possible d'écrire du code javascript à l'intérieur. Pour éviter des problèmes, on mettra nos balises <script> avant la fermeture de la balise <body>.

Chargement des scripts – fichier externe

```
● ● ●  
  
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <!-- [...] -->  
    <script src="chemin-vers-fichier.js" defer></script>  
    <!-- [...] -->  
  </head>  
  <!-- [...] -->
```

Notre code javascript est contenu dans un fichier externe puis chargé dans un fichier HTML. Le chemin peut être relatif ou absolu.

Attribut "defer"

```
<!-- [...] -->  
<script src="chemin-vers-fichier.js" defer></script>  
<!-- [...] -->
```

Source(s) :

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

Attribut "defer"

- Attribut propre à la balise `<script>`
- Indique au navigateur de charger le script **après chargement** des balises HTML
 - Limite les risques de bugs si le script est chargé avant les balises HTML
- Ne fonctionne qu'avec l'attribut "src"
- **Pensez à le mettre**

Source(s) :

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

javascript

- Langage faiblement typé :
 - Un entier peut devenir une chaîne de caractères ! Et vice-versa.

Langage extrêmement permissif. Soyez rigoureux dans votre code !

Variables

- Non typées
 - Elles peuvent changer de type
- Permettent de contenir des valeurs
- Préférer un nommage clair et descriptif
 - Par convention, on utilise la camelCase.
Ex : jeSuisUneVariable

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 – Déclaration

- Deux mot-clés : "const" et "let"
- const : Déclare une constante
 - **Impossibilité** de réaffectation
- let : Déclare une variable qui peut muter
 - **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 – Déclaration



```
const jeSuisUneConstante = "Je ne changerai pas";  
let jeSuisUneVariable = "Je peux changer"  
  
/* A éviter - mot-clé var*/  
var onNeMUtilisePlus = "";
```

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 – Déclaration

Il existe également le mot-clé “var” pour déclarer des variables. Il est désuet et provoque des effets de bord (hissage – voir sources pour infos).

Préférez toujours “const” et “let”

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
- <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

Variables – Règles de nommage

- Ne pas mettre d'espaces
- Ne pas commencer par un nombre
 - Mais peut contenir un nombre
- Ne pas mettre de tirets ou autres caractères spéciaux (underscore possible)
- Éviter les accents

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 – Règles de nommage

```
const ma_var = 42;  
let formation2 = "MMI";  
const classesLycee = ["2nde", "1ere", "Tle"];
```



Autorisé

- camelCase
- Présence de nombre après le début
- Underscore

Interdit

- Espaces
- Commence par un nombre
- Tirets

```
const 8mauvaiseVar = 42;  
let form-ation = "MMI"  
const classes Lycee = ["2nde", "1ere", "Tle"]
```



Fonctions

- Permettent de réutiliser le code
 - Évite de se répéter
- Permettent de mieux séparer le code
 - Meilleure lisibilité
- Créent un contexte qui leur est propre
 - Toute variable créée à l'intérieur n'est pas accessible à l'extérieur

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Fonctions>
- https://fr.wikiversity.org/wiki/Introduction_g%C3%A9n%C3%A9rale_%C3%A0_la_programmation/Fonctions

Fonctions

- Contiennent un ensemble d'instructions
- **Peuvent** contenir une autre fonction
- **Peuvent** appeler une autre fonction
- **Peuvent** retourner une valeur

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions>
- https://fr.wikiversity.org/wiki/Introduction_g%C3%A9n%C3%A9rale_%C3%A0_la_programmation/Fonctions

Fonctions- Exemple de code



```
// Déclaration de la fonction
const maFonction = (parametre) => {
    let maVariable = "hello";

    return maVariable;
}

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

On définit une fonction qui s'appelle "maFonction" qui est appelée ensuite

Fonctions – mot-clé "return"

- Permet d'assigner le "résultat" d'une fonction dans une variable
- 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>

Fonctions – mot-clé "return"

- Utilisable **uniquement** dans une fonction
- Met fin à l'exécution d'une fonction
 - Toute ligne après le mot-clé "return" et au même niveau ne sera pas exécutée
- Disponible dans d'autres langages de programmation

Source(s) :

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

Fonctions – mot-clé "return"

```
const maFonction = (param1, param2) => {  
  return param1 + param2;  
  console.log("ne sera jamais exécuté");  
}
```

Cette ligne est après un "return" et au même niveau, elle ne sera jamais exécutée

Source(s) :

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

**Une fonction n'est pas obligée
d'avoir le mot-clé "return"**

Fonctions – Paramètres

- Définissent la signature d'une fonction
- **N'existent que dans la fonction**
- Valeurs définies lors de l'appel de la fonction
 - Valeurs appelée "arguments"
- Séparés par une virgule

Source(s) :

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

Fonctions – Paramètres

```
const soustraction = (param1, param2)
// Les valeurs param1 et param2
// n'existent que dans la fonction
return param1 - param2;
}

soustraction(16, 8);
```

Paramètres

Nom de fonction

Arguments

Source(s) :

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

Fonctions – Paramètres

```
const soustraction = (param1, param2)
// Les valeurs param1 et param2
// n'existent que dans la fonction
return param1 - param2;
}
```

```
soustraction(16, 8);
```

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

On appelle notre fonction les arguments 8 et 16. L'ordre des arguments sera le même dans la fonction :

- param1 = 16
- param2 = 8

Source(s) :

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

Fonctions – Valeurs par défaut

- Appliquent une valeur aux paramètres non définis



JS

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

Si l'argument "b" n'est pas défini lors de l'appel de la fonction. Alors sa valeur sera égale à l'entier "1"

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Functions/Default_parameters

Débugger son code

- Utilisation de la fonction : `console.log()`
 - Permet de debugger son code
 - Équivalent js de la fonction php "`print()`"
- Affichage du contenu de `console.log` dans 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>

Débugger son code



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

On utilise la méthode `console.log()` pour afficher le contenu d'une variable

Source(s) :

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

Pratiquons ! - Initiation javascript (Partie 1/2)

Pré-requis :

- Avoir la ressource ressources/initiation-javascript

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/s3-integration-web/travaux-pratiques/numero-1/s3-integration-web_travaux-pratiques_numero-1.ressources.zip

Les fonctions en résumé



```
// Définition de la fonction
const maFonction = (param) => {
  /* instructions */
  return param;
}

const monResultat = soustraction(42, 1337);
```

Le résultat d'une fonction peut être stocké dans une variable **si et seulement si** la fonction retourne quelque chose

Source(s) :

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

Point technique : Le point-virgule (;)

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

Trouver ses erreurs

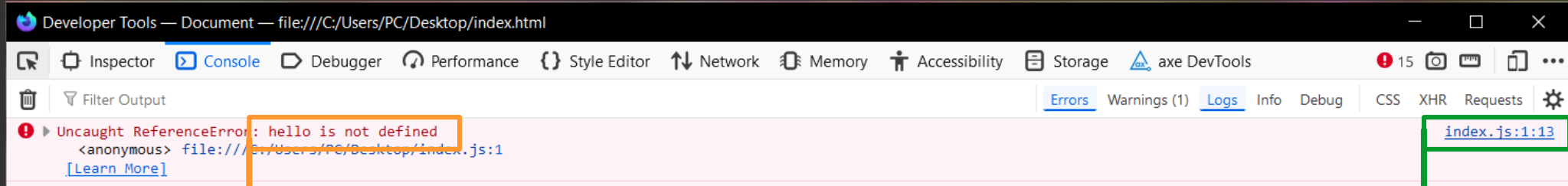
- Utilisation de la console du navigateur
 - Onglet “Console”
- Indique la ligne **exacte** du problème
- Message d'erreur en anglais

Pensez-y si vous avez un problème

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/What_went_wrong

Trouver ses erreurs



Message erreur
(ici la variable n'est pas définie)

Ligne et colonne de l'erreur
(cliquez dessus pour afficher le code
problématique dans le fichier)

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/What_went_wrong


**Pensez à la console du navigateur,
si votre code ne se comporte pas
correctement**

Source(s) :

- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/What_went_wrong

Conditions (if, else if, else)

- Permet de tester une condition et exécute son contenu 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`
- Utilisez tout le temps le triple égal 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 bloc (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');  
}
```

On teste la valeur d'une variable "maVariable"

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" ?

Conditions (if, else if, else) multiples

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

Source(s) :

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

Littéraux de gabarits

- Permet d'afficher une expression dans une chaîne de caractères
 - Expression : variable, résultat de fonction...
- Délimité par des backticks (`)
- Plus performant qu'une chaîne de caractères classique

Source(s) :

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

Littéraux de gabarits

- Plus performant qu'une chaîne de caractères classique
- Permet la gestion du texte multilignes
- Utilise la syntaxe `${expression}` pour interpréter la valeur

Source(s) :

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

Littéraux de gabarits



JS

```
const maVariable = "MMI";  
console.log(`Je suis en ${maVariable}`); // Je suis en MMI
```



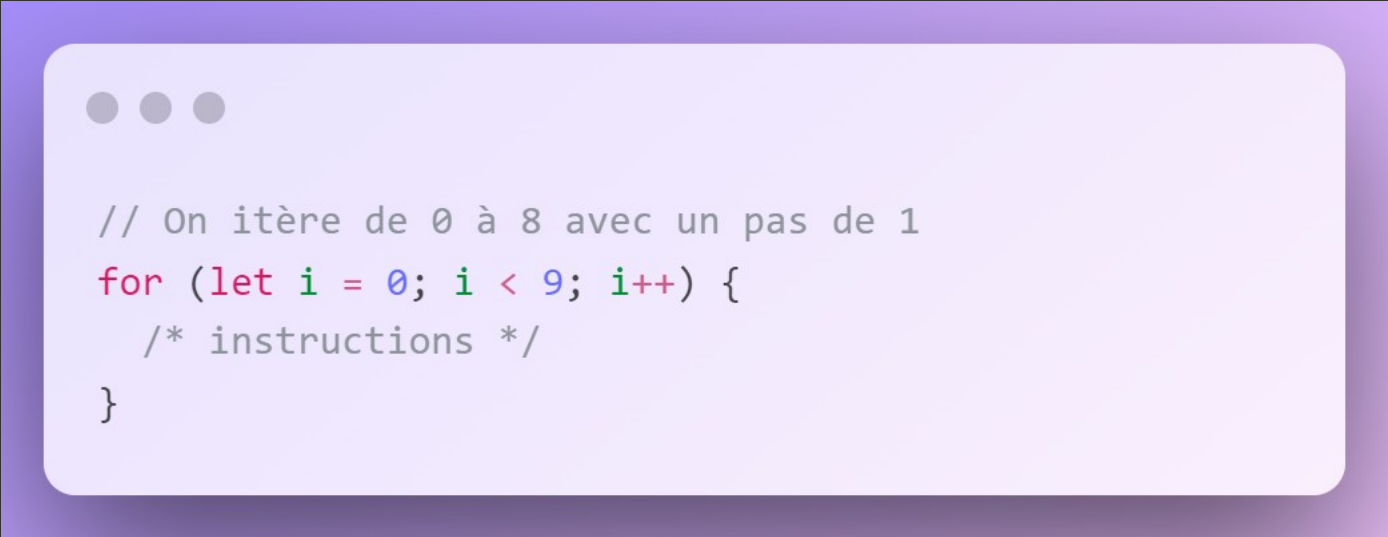
La variable sera compilée

Source(s) :

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

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 itérée 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

- Le pas d'une boucle peut être décimal
 - Ex : On peut faire des incréments de 0.1
- Il existe également les boucles while et do...while

Source(s) :

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

Pratiquons ! - Initiation javascript (Partie 3)

Pré-requis :

- Avoir la ressource `ressources/initiation-javascript`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/s3-integration-web/travaux-pratiques/numero-1/s3-integration-web_travaux-pratiques_numero-1.ressources.zip

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 → "click"
- Focus clavier → "focus"
- perte de focus clavier → "blur"
- survol → "mouseover"
- **Quelle précaution devons-nous prendre ?**
- 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>

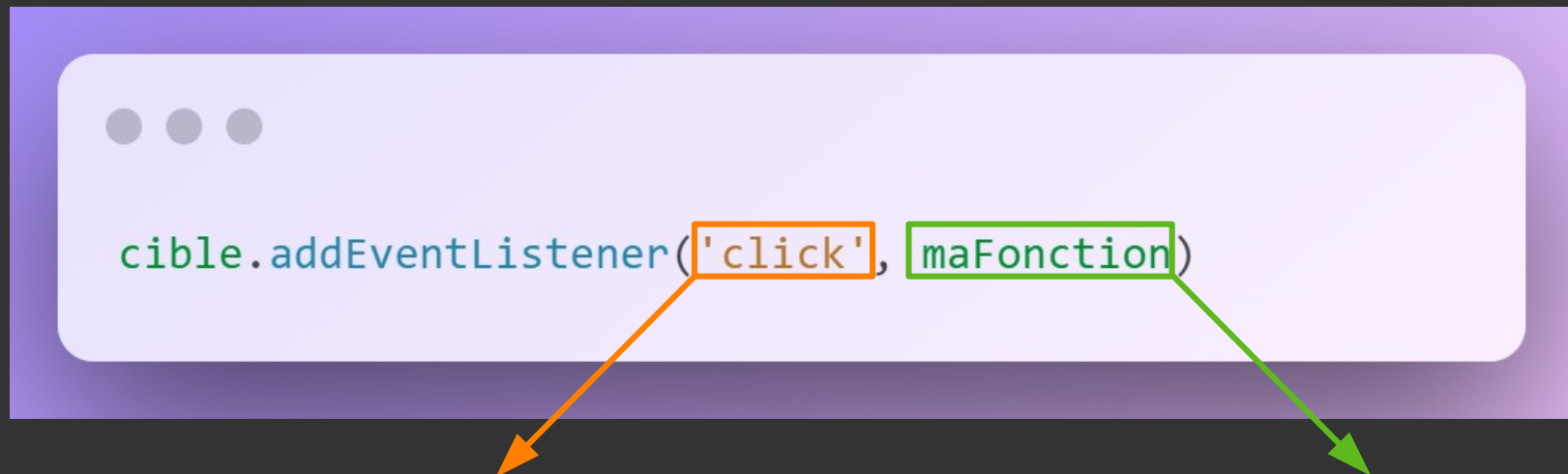
Évènements – Méthode "addEventListener"

- Permet d'écouter un évènement
- Deux arguments au minimum :
 - type d'évènement
 - fonction
 - **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 – Méthode "addEventListener"



Type d'évènement

Référence de la fonction à appeler
lorsque l'évènement est réalisé

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 (entre autres)

Source(s) :

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

Évènements – Fonction d'évènement

```
function maFonction(evt){  
    // Contient l'évènement et d'autres informations  
    console.log(evt);  
}  
  
// Retourne le premier <button> trouvé  
const cible = document.querySelector("button");  
// Applle la fonction "maFonction" au clic sur le bouton  
cible.addEventListener("click", maFonction);
```

La fonction “maFonction” sera appelée quand on clique sur un bouton

Évènements – Cible d'un évènement

- Méthodes pour récupérer les éléments :
 - `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>

Évènements – Cible d'un évènement

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



JS

```
// 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 l'évènement sur chacun des éléments

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)
}
```

Sur chacun des éléments, on applique un évènement de type “click”

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**
- Mettre l'évènement sur les éléments interactifs (bouton, champ...) pour ne pas exclure les utilisateurs au clavier

Évènements – Attribut HTML

- Possibilité de lier un évènement via un attribut html



JS

```
<button onClick="maFonction()">Mon bouton</button>
<script>
  const maFonction = () => {
    alert("ok");
  }
</script>
```

Cette méthode est à proscrire

Évènements – Attribut HTML

- Inconvénients :
 - Nuit à la lisibilité du code (comme l'attribut style en CSS)
 - Duplication de code

Évènements – Attribut HTML



JS

```
<button onClick="maFonction()">Mon bouton</button>
<script>
  const maFonction = () => {
    alert("ok");
  }
</script>
```



```
<button class="my-classe">Mon bouton</button>
<script>
  const cible = document.querySelector(".ma-classe");
  cible.removeEventListener("click", () => {
    alert("ok");
  });
</script>
```



Pratiquons ! - Initiation javascript (Partie 4)

Pré-requis :

- Avoir la ressource `ressources/initiation-javascript`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/s3-integration-web/travaux-pratiques/numero-1/s3-integration-web_travaux-pratiques_numero-1.ressources.zip

Évènements – Suppression d'un évènement

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



JS

```
const cible = document.querySelector(".ma-classe");  
// Retire l'appel de la fonction "maFonction" au clic  
cible.removeEventListener("click", maFonction);
```

Source(s) :

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

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 doivent 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 et passer des informations au javascript
- classes CSS : styliser un élément
- id : pour les ancrs et éléments de formulaires

Pratiquons ! - Initiation javascript (Partie 5/6)

Pré-requis :

- Avoir la ressource `ressources/initiation-javascript`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/s3-integration-web/travaux-pratiques/numero-1/s3-integration-web_travaux-pratiques_numero-1.ressources.zip

Point technique : Le DOM

- Signifie Document Object Model
- Représente toutes les balises de la page sous forme d'objets :
 - document (js) → page entière (html)
 - *Écrivez "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 / supprimer :
 - Les attributs d'une balise HTML
 - Les balises HTML de la page

Source(s) :

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

Point technique : Le DOM



```

```



```
// 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.avif";
```

typescript

- “Super” javascript développé par Microsoft
- Améliore la sécurité et la qualité du code
 - Les variables, fonctions... sont typées
- Inspiré par le langage C# (abordé en S4)
- Nécessite NodeJS pour fonctionner
 - Car le typescript est compilé en js à la fin

Source(s) :

- <https://www.typescriptlang.org/>
- <https://www.typescriptlang.org/play/> - Pour tester en ligne

Questions ?

