jQuery

Table des matières

1. Introduction

jQuery est une librairie, plus ou moins comme Bootstrap, qui permet d'écrire du code JavaScript de manière plus rapide / pratique. Il consiste en une série de fonctions pré-écrites que nous allons pouvoir utiliser telles-quelles, et possède sa propre syntaxe. Par exemple :

Sélectionner un élément HTML :

```
<div id="maDiv">
    Bonjour
</div>
<script src="https://code.jquery.com/jquery-3.7.1.js" integ
<script>
    // Javascript ici :)
    document.getElementById('maDiv').innerHTML = 'Salut';
    // jQuery ici :)
    $('#maDiv').html('Salut !')
</script>
```

Il fonctionne sur une logique de **sélecteurs**, qui sont les mêmes qu'en CSS, c'est-à-dire les balises html, le point **.** pour les classes et le dièse **#** pour les id. Et pour l'intégrer on utilise comme avec Bootstrap, un **CDN**.

https://releases.jquery.com/

2. Mode d'emploi

On se sert des sélecteurs pour **cibler** un élément HTML, puis on lui applique une **fonction.** Ici par exemple on applique la fonction **click()** sur le bouton en ciblant son id, puis on applique la fonction **hide()** sur la div avec l'id **rouge**.

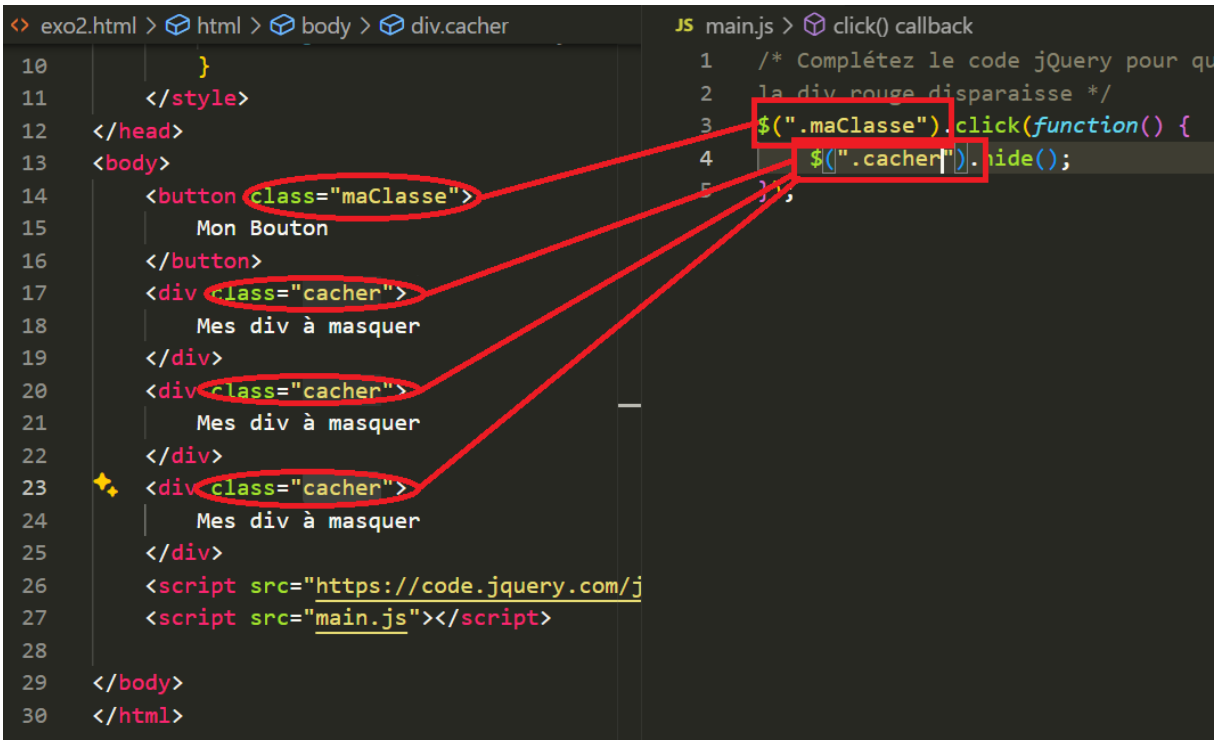
exo2.html > html

```
10          }
11      </style>
12  </head>
13  <body>
14      <button id="monBouton">
15          Mon Bouton
16      </button>
17      <div id="rouge">
18          Ma div à masquer
19      </div>
20      <script src="https://code.jquery.com/j
21      <script src="main.js"></script>
22
23  </body>
```

main.js > ...

```
1   /* Complétez le code jQuery pour que
2   la div rouge disparaisse */
3   $("#monBouton").click(function() {
4       $("#rouge").hide();
5   });
```

On peut également cibler avec la classe, comme ici, que l'on veuille cibler un élément ou plusieurs :



Enfin, on peut cibler un élément par sa balise, et éventuellement ses attributs en passant l'attribut en question dans deux crochet **[]** et en donnant leur valeur avec un **=**. Cela nous donne le sélecteur suivant : « balise[attribut='valeur'] », comme ici :

3. Syntaxe

jQuery utilise le signe **$** en guise d'appel. À chaque fois que vous verrez le signe **$** dans du JavaScript, cela veut dire que c'est à jQuery qu'on fait appel. C'est là toute la subtilité de jQuery : ce n'est pas un nouveau langage en soi, bien qu'il possède sa propre syntaxe, mais il coexiste avec JavaScript !

Quelques exemples :

```
// Javascript ici :)
document.getElementById('maDiv').addEventListener('click', function() {
    alert('Hello');
});
// jQuery ici :)
$('#maDiv').click(function() {
    alert('Hello');
});

// Vérifier si un de mes élements a bien une classe :
document.getElementById('maDiv').classList.contains('maClasse');
$('#maDiv').hasClass('maClasse');
```

```
// Afficher ou masquer un élément :
document.getElementById('maDiv').style.display = 'none';
$('#maDiv').hide();
document.getElementById('maDiv').style.display = 'block';
$('#maDiv').show();
```

```
// Gérer le hover sur un élément :
$('#maDiv').hover(function(){
    // Faire un truc au survol
}, function() {
    // Faire un truc quand on sort
});
```

4. Fonctions utiles

- **.click**(function() { /* Action à effectuer * /}) ; permet de gérer un **évènement** (le click)
- **.change**(function() { /* Action à effectuer * /}) ; permet de gérer un **évènement** (le change)
- **.on('evenement',** function() { /* Action à effectuer * /}) sera la façon préférable de déclarer les eventListener. En effet, un évènement déclaré avec **on()** sera appliqué même si l'élément cible arrive **après** le chargement du DOM, ce qui ne sera pas le cas avec les fonctions évènements « classiques » comme **.click** ou **.change**.
  Pour qu'un évènement soit détecté sur un élément chargé **après**, il faut utiliser la syntaxe suivante :

```
$(document).on("monEvenement(click, change, mouseenter...)",
"monSelecteur(#monId, .maClasse...)", function () {
    // Code à exécuter
}) ;
```

Exemple :

```
$(document).on('click', '.maDiv', function() {
    // Code à exécuter
});
```

- **.html**(/* Contenu à écrire */) ; permet d'écrire dans l'élément cible. Si on ne lui passe pas d'argument, html() nous renvoie le contenu **déjà présent**.
- **.data**(/* nom de ma data */) ; permet d'accéder à l'attribut data que l'on a défini sur l'élément cible
- **.each**(function() { /* Action à effectuer * /}) ; permet d'initier une **boucle** sur un tableau. À chaque itération, on aura accès au sélecteur **$(this)**, qui correspondra à l'itération en cours
- **.addClass('classeAAjouter') / .removeClass('classeAEnlever') ;** permet d'ajouter ou d'enlever une classe à notre élément. Pratique pour rapidement ajouter ou enlever des propriétés CSS ! Si on ne passe aucun paramètre à **removeClass()**, alors **toutes** les classes sont supprimées
- **.toggleClass('classe')** permet de gérer l'ajout ou la suppression d'une classe sans devoir vérifier si elle existe déjà ou non.
- **.hover**(function() { /* Action à effectuer quand on entre dans la zone de survol * /}, function() { /* Action à effectuer quand on quitte la zone de survol * /})
- **.animate** permet de gérer une animation sur une propriété. Par exemple, on peut utiliser animate sur la propriété **scrollTop** afin d'emmener la barre de scroll sur un élément précis.
- **offset** permet d'accéder à la position d'un élément, avec offset().top et offset().left
- **height()** permet d'obtenir la taille d'un élément
- **width()** permet d'obtenir la largeur d'un évènement
- **attr()** permet d'accéder aux attributs HTML comme « alt », « src », « id »…
- **remove()** permet de supprimer un élément purement et simplement. Il n'est pas **masqué**, il n'existe plus
- **after()** et **before()** permettent d'ajouter des éléments **après** ou **avant** notre cible. Par exemple, on peut dire $('#maDiv').after(« Lorem ipsum ») et la balise <p< sera affichée **après** la div
- **append()** et **prepend()** permettent également d'ajouter du contenu, mais cette fois à **l'intérieur** de l'élément, respectivement à la fin et au début.
- **css('propriété CSS', 'valeur de la propriété')** permet de modifier une propriété CSS en lui passant la nouvelle valeur de la propriété ciblée
- **empty()** permet de vider un élément de son contenu
- **find()** permet de trouver un élément DANS notre élément cible. Par exemple trouver toutes les balises p dans une div
- **trim()** permet de supprimer les espaces au début et à la fin de la chaîne de caractère. Attention, trim a deux syntaxes, on peut le noter comme ceci $('input').val().**trim()**, mais trim peut aussi se noter ainsi : **$.trim(**maChaineOuMaVariable**)**
- **parent()** permet de remonter le DOM d'un cran pour sélectionner l'élément **contenant.** Par exemple si on a un bouton dans un <li>, on va pouvoir cibler le li en cliquant sur le bouton comme ceci :

```
$(document).on('click', '.supprimer', function() {
    $(this).parent().remove();
});
```

TESTER UNE TOUCHE LORS D'UN ÉVÈNEMENT KEYPRESS

Il peut arriver que l'on ait besoin de vérifier quelle touche a été tapée avant de lancer le traitement. Pour cela, on peut utiliser la syntaxe suivante, ou le paramètre ***e*** de la fonction sera pré-rempli par défaut et nous permettra d'accéder à **which**, pour tester le code de la touche pressée. Le cas d'école est celui de la touche entrée, que l'on testera comme ceci :

```
        $('#tache').on('keypress', function(e) {
            if(e.which == 13) {
                ajouterTache();
            }
        });
```