jQuery

SYNTHESE DE L'ESSENTIEL DE JQUERY

Léo LAROU-CHALOT
IPSSI | 25 RUE CLAUDE TILLIER, 75012 PARIS

Introduction

jQuery est une bibliothèque JavaScript, permettant la simplification et l'accélération de l'écriture de nos script JavaScript.

Ce cours, synthèse du cours « L'essentiel de jQuery » suivi sur LinkedIn Learning à pour objectif de servir de mémo et de référentiel pour la suite de mon cursus en informatique.

Objectifs

- Accélérer la prise en main de jQuery
- Familiarisation avec la documentation du projet jQuery
- Montrer quelques exemples de l'utilisation de JavaScript dans un site web

Qu'est-ce que jQuery ?

jQuery est ce que l'on pourrait nommer une bibliothèque de fonctions courantes.

« jQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture du script côté client dans le code HTML des pages Web. »
(Wikipédia)

Principales fonctionnalités

Parmi les fonctionnalités de jQuery, nous retrouverons :

- Sélectionner et manipuler les éléments de notre page web
- Créer, modifier, supprimer, etc. les éléments de notre page
- Manipuler les styles CSS, animer notre page
- Utiliser la programmation asynchrone avec AJAX
- Gérer les événements du JavaScript

Historique de jQuery

La première version de jQuery a été développée par John Resig en 2006

jQuery devient la bibliothèque JS la plus populaire du monde (2019 : utilisée par près 80% du million de site les plus populaire d'Internet)

Sa popularité est en légère baisse depuis les apparitions des bibliothèque Angular ou React.

jQuery a conduit à l'intégration de certaines techniques popularisées par elle-même dans le langage JavaScript lui-même ! (querySelector ou querySelectorAll par exemple)

FAQ jQuery

Peut-on créer un site sans jQuery ? Oui ! Mais jQuery rend les choses plus faciles.

Peut-on mélanger du jQuery avec du JavaScript Vanilla ? Oui, les deux outils sont du JavaScript.

Peut-on utiliser jQuery avec d'autres bibliothèques ? Oui ! Mais attention aux possibles recoupement et confusions !

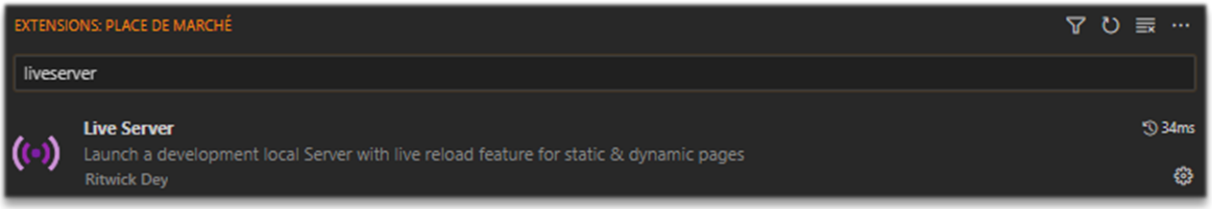
Peut-on étendre les capacités de jQuery ? Oui ! jQuery peut être étendu à l'aide de plug-ins.

Chapitre 1 : Prendre en main jQuery

Installer les outils préalables

Pour les besoins de ce cours, et par habitudes, j'utiliserai Chrome et ses outils de développement, et Visual Studio Code comme éditeur de code.

Nous utiliserons aussi les extensions Live Server pour visualiser l'avancée de notre projet en temps réel



Il faudra aussi télécharger jQuery. Pour ce faire : https://jquery.com

Télécharger le fichier

Nous aurons le choix entre la version complète :

ou la version « slim » .

Cette dernière exclu les utilisations des modules AJAX et des modules d'effets.

Le code affiché lorsque l'on clic sur l'un de ces liens sera à enregistrer, et à placer dans notre répertoire de travail, et à lier dans notre fichier HTML.

Utiliser un CDN

Un Content Distribution Network ou Réseau de Distribution de Contenu est un fichier hébergé sur un autre serveur.

https://code.jquery.com

jQuery 3.x

- jQuery Core 3.6.3 - uncompressed, minified, slim, slim minified

```
<script
  src="https://code.jquery.com/jquery-3.6.3.min.js"
  integrity="sha256-pvPw+upLPUjgMXY0G+8O0xUf+/Im1MZjXxxgOcBQBXU="
  crossorigin="anonymous"></script>
```

```
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script
        src="https://code.jquery.com/jquery-3.6.3.min.js"
        integrity="sha256-pvPw+upLPUjgMXY0G+8O0xUf+/Im1MZjXxxgOcBQBXU="
        crossorigin="anonymous"
    ></script>
    <title>Installer jQuery</title>
</head>
```

On remarque que l'attribut « src » pointe vers un site, et plus vers le fichier jQuery précédemment téléchargé.

Avantages

Cela fait moins de fichiers à télécharger et à héberger.

Les CDN utilisent des infrastructures très performantes, dites « redondantes », cela signifie qu'il y a des très nombreux serveurs CDN répartis à travers le monde. De se fait, peut importe la localisation de l'internaute, un serveur sera très probablement situé à côté. C'est un gage de performance !

Un autre avantage, est que si l'un de ses serveurs tombe en panne, les autres pourront prendre le relais sur la distribution de contenus.

L'avantage le plus important est que de nombreux sites utilisent les CDN, ainsi, si l'internaute à visité un site utilisant le même CDN que le nôtre, il n'aura même plus à télécharger la bibliothèque jQuery lors de sa visite sur notre site puisqu'elle sera déjà présente dans le cache de son ordinateur !

Gain de bande passante, et de performances !

Utiliser la documentation jQuery

La documentation (disponible uniquement en anglais) est disponible sur le site : jQuery API Documentation.

Cette documentation présente l'ensemble des fonctionnalités propres à jQuery, mais précise également la liste des fonctions dites « dépréciées ». Une fonction « dépréciée » est une fonction qui s'est vue retirée de la bibliothèque.

Deux raisons à cela :

- La fonction n'était pas suffisamment utilisée
- La fonction a été remplacée par une autre fonction plus optimisée

Attention : Lors du suivi de tutoriels sur internet, il est possible de tomber sur l'utilisation de ces fonctions « dépréciées ». Il faudra donc être vigilant à cela pour éviter tout problème lors de l'exécution de nos script.

Une barre de recherche est également disponible pour trouver de la documentation sur un concept ou une fonction en particulier.

Par exemple, si l'on a besoin d'informations sur la fonction « slideToggle() »

La documentation nous permet d'accéder aux détails de la fonction, de ses paramètres, et plus globalement, de son utilisation.

Dans cet exemple, les paramètres doivent être séparés par une virgules et sont entre « crochets ». Cela signifie qu'ils ne sont pas obligatoires.

Chapitre 2 : Utiliser les sélecteurs et les filtres

Créer un objet jQuery

Pour bénéficier des fonctionnalités que nous propose jQuery, il faudra commencer par créer un objet jQuery.

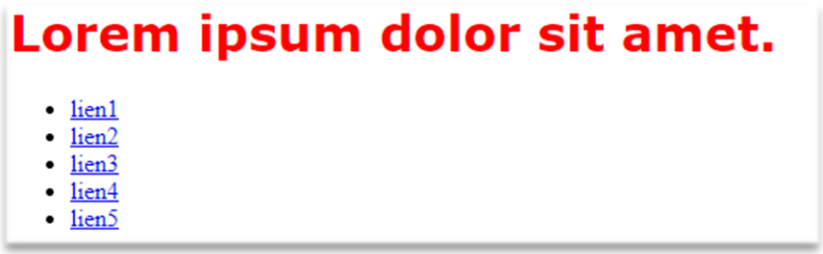
Syntaxe de base

```
$(document).ready(() => {
  jQuery('h1').css({fontFamily:'Verdana',color:'red'});
});
```

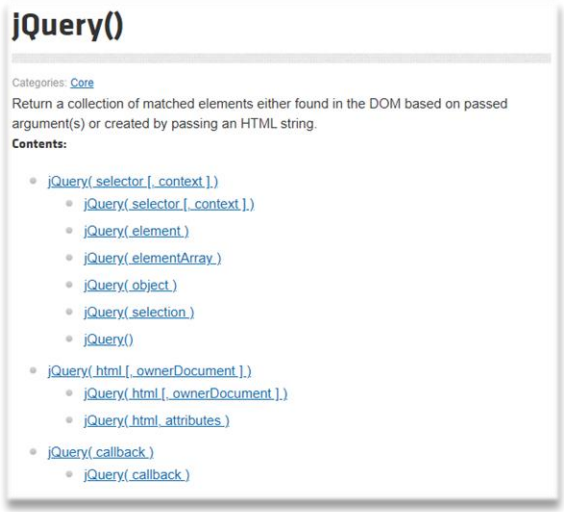
Cette syntaxe indique que l'on cible tous les éléments « h1 » de notre code HTML, et que l'on définit leur « fontFamily » et leur « color » à « Verdana » et à « red ».

Attention cependant, de même qu'en JavaScript l'écriture des propriétés CSS se fera en utilisant l'écriture « CamelCase » (une minuscule au début, puis une majuscule à chaque nouveau terme).

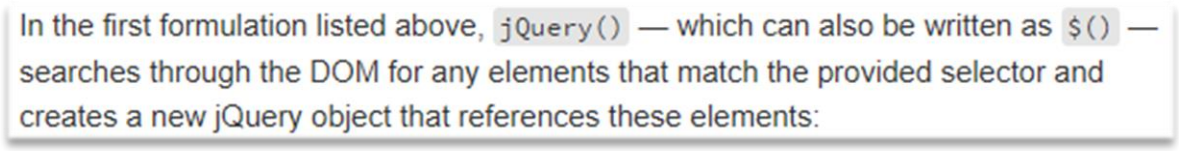
```
<header id="mainHeader">
    <h1>Lorem ipsum dolor sit amet.</h1>
    <nav id="mainNav">
        <ul>
            <li><a href="#">lien1</a></li>
            <li><a href="#">lien2</a></li>
            <li><a href="#">lien3</a></li>
            <li><a href="#">lien4</a></li>
            <li><a href="#">lien5</a></li>
        </ul>
    </nav>
</header>
```



Jetons un œil à la documentation de la fonction jQuery.



Nous voyons qu'il existe une multitude de façon d'utiliser cette fonction. En allant un peu plus bas :



Il est dit : Dans la première formulation répertoriée ci-dessus, **jQuery()** — qui peut également être écrit comme **$()** — recherche dans le DOM tous les éléments qui correspondent au sélecteur fourni et crée un nouvel objet jQuery qui fait référence à ces éléments.

Ainsi :

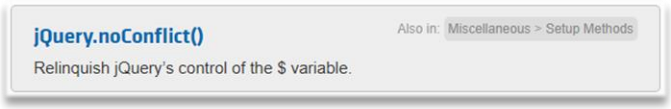
Peut s'écrire de la façon suivante :

L'alias « $ » et son utilisation

On dira que le « $ » est un **alias** de la fonction « jQuery ».

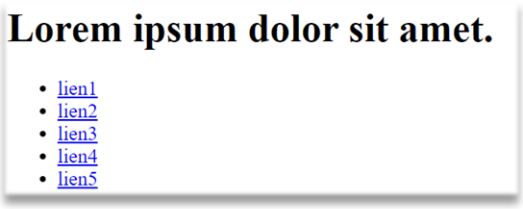
Dans notre métier de développeur, rappelons que plus le code est court et lisible, mieux l'on se porte, les alias proposés par jQuery sont donc très utiles en ce qui concerne l'écriture de notre code.

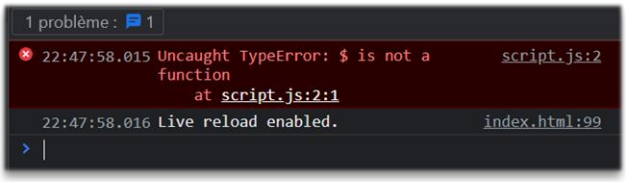
Attention d'autres bibliothèques utilisent ce même procédé et cela peut entrainer des confusions. Pas de panique, la bibliothèque jQuery propose une solution. Si l'on va dans la section « core » de la documentation, nous pouvons trouver la fonction « jQuery.noConflict() ». Cette fonction permet de désactiver les alias « $ ».



Cette fonction sera à mettre à la première ligne de notre fichier JavaScript.

Ainsi, l'aperçu navigateur deviendra :





Et pour cause, l'utilisation de « $ » n'est donc plus reconnu, il faudra donc utiliser la syntaxe « jQuery() » en toutes lettres.

Attendre le chargement complet de la page

Et si, au lieu de positionner notre balise script en bas de page HTML, nous le placions en la balise « head » comme ceci :

Il est important de noter que le fichier est appelé après l'utilisation du CDN, car nous utilisons des éléments jQuery proposé par celui-ci.

Le navigateur n'interprètera pas nos modifications précisées dans notre « script.js ».

Etrange, il n'y a pourtant pas d'erreur dans notre console.

Explications :

Le navigateur va charger nos fichier de haut en bas. Or, au moment de charger notre fichier « script.js », il n'aura pas encore chargé le contenu de notre balise « body ».

Pour résoudre ce problème, il faudra penser à préciser au navigateur d'attendre le chargement complet de notre page pour charger notre fichier « script.js » en utilisant la directive « defer ».

Ou alors, directement dans notre fichier « script.js », nous pourrons décider d'utiliser l'évènement « ready() » pour préciser d'attendre le chargement complet du DOM.

Il faudra préciser la fonction contenant notre code à exécuter après ce chargement.

La plupart des scripts jQuery commencent par cette ligne de code.

Utiliser les sélecteurs CSS

L'utilisation des sélecteurs CSS en JavaScript est une étape obligatoire pour la gestion de notre site.

Dans cette partie, nous verrons les méthodes que nous propose jQuery (l'une des plus grosses innovations de jQuery) par rapport à JavaScript Vanilla.

Avant jQuery il était parfois compliqué de cibler des éléments précis dans notre page. Grâce à jQuery, ce processus est devenu extrêmement simple, et pour cause, jQuery utilise les mêmes techniques de sélections que le CSS. C'est d'ailleurs de là que la bibliothèque jQuery tire son nom.

Par exemple :

En JavaScript Vanilla	En jQuery
	

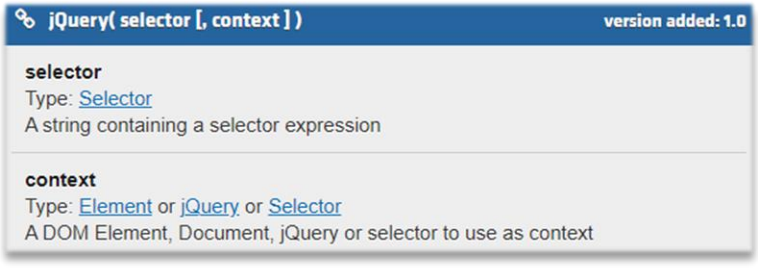
De la même façon en utilisant les sélecteurs CSS :

Ainsi, l'utilisation des sélecteurs contextuels, et multiples est tout à fait possible avec jQuery !

On remarquera la proximité avec « document.querySelector() » et « document.querySelectorAll() », c'est en fait jQuery qui a popularisé cette façon de cibler les éléments, en effet, ces fonctions n'étaient pas natives à JavaScript au moment de l'apparition de jQuery !

Sélectionner dans un contexte bien précis

La fonction « jQuery() », utilisable avec l'alias « $ » comme nous l'utilisons peut être développée.
En allant sur la documentation, nous trouvons :



Nous voyons qu'un paramètre **[context]** peut être passé comme argument dans notre fonction.

Il s'agit simplement d'un autre type de syntaxe pour préciser l'élément ciblé avec jQuery.

Ainsi :

Peut s'écrire :

Les deux syntaxes ciblent le même élément (éléments « h1 » situés dans les éléments « article »).

Mais à quoi cela peut-il servir ?

Cela peut permettre par exemple de sélectionner les éléments « article » dans une variable, et de passer cette variable comme contexte dans la fonction jQuery (« $ »)

L'intérêt de cette syntaxe est que, dans nos scripts jQuery, nous ne saurons pas toujours à l'avance dans quel(s) contexte(s) sélectionner nos éléments. Le contexte pourra par exemple dépendre de la page sur laquelle on se trouve, du bouton sur lequel on clic, ou même de la langue du site.

Utiliser les filtres du CSS

Comme vu précédemment, nous pouvons voir que le ciblage des éléments via les filtres CSS est tout à fait faisable, et extrêmement simplifié grâce à jQuery.

```
$(document).ready(() => {
    let allArticles = $("article");
    $("h1", allArticles).css({ fontFamily: "Verdana", color: "red" });

    // ? Sélection des premiers "p" au sein des différents conteneurs
    $("p:first-of-type").css({ fontFamily: "Verdana", color: "green" });

    // ? Sélection des "p" qui sont le premier enfant de leur conteneur
    $("p:first-child").css({ fontFamily: "Verdana", color: "blue" });

    // ? Sélection des "p" qui sont le dernier enfant de leur conteneur
    $("p:last-child").css({ fontFamily: "Verdana", color: "blue" });

    // ? Sélection des "p" qui sont le 3ème enfant de leur conteneur
    $("p:nth-child(3)").css({ fontFamily: "Verdana", color: "orange" });

    // ? Sélection du 3ème élément "p" de chaque conteneur
    $("p:nth-of-type(3)").css({ fontFamily: "Verdana", color: "yellow" });

});
```

Connaître les extensions de jQuery

En plus des sélecteurs CSS, jQuery propose des sélecteurs supplémentaires. Comme ils sont disponible directement dans la bibliothèques jQuery, ils sont disponibles en jQuery mais ne fonctionneront pas en CSS.

:gt (greater than)

```
$('p:gt(1)').css({fontFamily: 'Verdana', color: 'red'});
```

Ici je sélection tous les éléments « p » plus grands que « 1 ». Qu'est-ce que cela signifie ?

Commençons par rappeler qu'il s'agit d'un sélecteur propre à jQuery, et que jQuery est une bibliothèque JavaScript, et qu'en JavaScript, quand on compte, on commence par « 0 ».

Ainsi cela revient à compter les paragraphes en partant de « 0 », tous ceux situés à un index plus grand que « 1 » seront affectés par la sélection !

:lt (less than)

```
$('p:lt(1)').css({fontFamily: 'Verdana', color: 'red'});
```

En suivant cette logique, le sélecteur « lt » ciblera tous les paragraphes dont l'index est inférieur à « 1 ».

:has

```
4    $('article:has(p.link)').css({fontFamily:'Verdana',color:'red'});
```

« has », du verbe « have » en anglais, qui veut dire « avoir ».

Le sélecteur « :has() » permet de spécifier une condition de sélection.

Ici, je souhaite sélectionner tous les « article » qui possèdent un « p » avec la classe « .link »

:header

```
4    $(':header').css({fontFamily:'Verdana',color:'red'});
```

Le sélecteur « :header » va permettre la sélection de tous les titres, et ce peu importe leur niveau (« h1 » jusqu'à « h6 »).

Inconvénients

Comme ces sélecteurs sont propres à jQuery, nous ne pourrons pas utiliser les performances natives de l'interpréteur CSS du navigateur pour aller sélectionner ces éléments dans la page.

Ces sélecteurs ne s'appliquent qu'au JavaScript et cela implique des performances un petit peu moindre au niveau de ces sélecteurs.

En savoir plus

Pour le complément sur les sélecteurs jQuery, rendez-vous ici : jQuery Extensions | jQuery API Documentation.

Utiliser la fonction « filter() »

Cette fonction permet de filtrer une sélection, c'est-à-dire de retirer des éléments d'une sélection initiale pour ne garder que les éléments qui satisfont à un critère supplémentaire.

Documentation : .filter() | jQuery API Documentation

Sélection initiale :

```
4    $('p').css({fontFamily:'Verdana',color:'red'});
```

Avec le filtre :

```
4    $('p').filter(':not(.link)').css({fontFamily:'Verdana',color:'red'});
```

Ici on spécifie la sélection de tous les éléments « p » qui ne contiennent pas la classe « .link ».
Le sélecteur « :not » existe en CSS, mais il a été rajouté à la suite de son utilisation avec jQuery.

Il est possible, comme vu précédemment, de stocker notre sélection dans une variable :

```
3    let allLinks = $(':not(.link)');
4    $('p').filter(allLinks).css({fontFamily:'Verdana',color:'red'});
```

Chapitre 3 : Manipuler les éléments du DOM

Présentation du poste de travail


```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>N&B photographies - À propos</title>
    <link rel="stylesheet" href="css/normalize.css">
    <link rel="stylesheet" href="css/nb.css">
    <script src="js/jquery-3.5.1.min.js"></script>
    <script src="js/randomImage.js" defer></script>
    <script src="js/footnotes.js"></script>
</head>
```

Nous utiliserons le support de cette formation pour travailler avec jQuery.

Dans ce chapitre, nous allons travailler avec les attributs « data- ».

De quoi s'agit-il ?L'attribut « data-« est une nouveauté de HTML5 qui nous permet d'associer des données aux balises HTML.

```
<span data-footnote="Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae;">repudiandae</span>
```

Pour que cela fonctionne, il est obligatoire que le nom de l'attribut commence par « data- » suivi du nom de notre attribut. Cette syntaxe va nous permettre de lier la valeur de l'attribut au texte encadré par la balise « span ».

Mettre en place le script

Le début de notre script pour travailler sur les éléments mentionnés ci-dessus se présentera de cette façon :

```
$(document).ready(() => {
  let footnotes = $('span[data-footnotes]');
  if(footnotes){

  }
});
```

Utiliser la fonction each()

Documentation : [.each() | jQuery API Documentation]

La fonction « .each » marche de la façon suivante :

```
$(document).ready(function(){
  let footnotes = $('span[data-footnote]');
  console.log(footnotes);
  if(footnotes){
    footnotes.each(function(index, element) {
      let current = $(element);
      console.log(current);
    });
  }
});
```

Elle prend en argument une fonction, qui elle-même aura deux paramètres : « index » et « element ».

Utiliser la fonction data()

Documentation : .data() | jQuery API Documentation

La fonction « data() » va nous retourner le contenu de l'attribut « data- » HTML que l'on va cibler :

```
$(document).ready(function(){
  let footnotes = $('span[data-footnote]');
  console.log(footnotes);
  if(footnotes){
    footnotes.each(function(index, element) {
      let current = $(element);
      let footnoteText = current.data('footnote');
      console.log(footnoteText);
    });
  }
});
```

```
20:44:54.212 footnotes.js:3
▸ S.fn.init(3) [span, span, span, prevObject: S.fn.init(1)]
20:44:54.213 Donec dictum blandit interdum. Vivamus eleifend massa neque, eu pharetra sapien venenatis a. footnotes.js:8
20:44:54.213 Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; footnotes.js:8
20:44:54.213 Quisque commodo eu turpis vel pharetra. Curabitur iaculis risus eget finibus fermentum. footnotes.js:8
```

Et voilà le résultat ! J'utilise l'objet jQuery « current » sur lequel nous bouclons, pour récupérer la valeur de l'attribut « data- » ciblé (ici, « footnote »).

Modifier le CSS avec jQuery

Documentation : .css() | jQuery API Documentation

La fonction « .css() » peut avoir deux rôles distincts :

- Getter : sous la forme « .css(propertyName) », cela va nous retourner la valeur de la propriété css ciblée
- Setter : sous la forme « .css(propertyName, value) », cela va nous permettre d'assigner une valeur à la propriété css ciblée.

Ici nous utiliserons la deuxième forme, la forme « setter »

Ainsi :

```
$(document).ready(function(){
  let footnotes = $('span[data-footnote]');
  console.log(footnotes);
  if(footnotes){
    footnotes.each(function(index, element) {
      let current = $(element);
      let footnoteText = current.data('footnote');
      current.css('text-decoration', 'underline dashed');
    });
  }
});
```

Sapiente ad nihil quam

Et voilà ! De cette façon nous avons pu modifier la propriété CSS « text-decoration » pour lui attribuer la valeur « underline dashed ».

Utiliser la fonction after()

Documentation : .after() | jQuery API Documentation

La fonction « .after() » va nous permettre d'ajouter du contenu **après** l'élément ciblé.

Elle pourra prendre en paramètre une chaîne HTML (un morceau de code HTML donc).

Ainsi :

```
$(document).ready(function(){
  let footnotes = $('span[data-footnote]');
  console.log(footnotes);
  if(footnotes){
    footnotes.each(function(index, element) {
      let current = $(element);
      let footnoteText = current.data('footnote');
      current.css('text-decoration', 'underline dashed').after(`<a href="#footnote-${index+1}" id="footnote-anchor-${index+1}"><sup>[${index+1}]</sup></a>`);
    });
  }
});
```

De cette façon, nous allons rajouter un indice entre crochets, lui-même contenu dans une balise « a » qui nous renverra vers notre bas de page, en ciblant « #footnote-{index+1} ».

Si la ligne de code vous parait trop longue, sachez qu'il est possible de scinder cette ligne de la façon suivante :

```
current
.css('text-decoration', 'underline dashed')
.after(`<a href="#footnote-${index+1}" id="footnote-anchor-${index+1}"><sup>[${index+1}]</sup></a>`);
```

Générer les éléments à la volée

Pour utiliser ces notes de bas de page, nous allons avoir besoin d'une liste numérotée (en HTML), et des éléments <li>.
Pour cela, nous allons de nouveau pouvoir utiliser la fonction jQuery(), mais d'une autre façon.

```
$(document).ready(function(){
  let footnotes = $('span[data-footnote]');
  console.log(footnotes);
  if(footnotes){

    let footnoteDisplay = $('<ol></ol>');

    footnotes.each(function(index, element) {
      let current = $(element);
      let footnoteText = current.data('footnote');

      current
      .css('text-decoration', 'underline dashed')
      .after(`<a href="#footnote-${index+1}" id="footnote-anchor-${index+1}"><sup>[${index+1}]</sup></a>`);

      $(`<li id="footnote-${index+1}">${footnoteText}</li>`);
    });
  }
});
```

Utiliser les fonctions append() et appendTo()

Nous avons vu précédemment comment créer nos éléments mais nous ne les avons pas encore injectés dans notre code HTML, c'est là qu'interviennent les fonctions « .append() » et « .appendTo() ».

Documentation : [.append() | jQuery API Documentation]

Documentation : [.appendTo() | jQuery API Documentation]

De cette façon, nous allons avoir le code suivant :

```
let footnoteDisplay = $('<ol></ol>');

footnotes.each(function(index, element) {
  let current = $(element);
  let footnoteText = current.data('footnote');

  current
  .css('text-decoration', 'underline dashed')
  .after(`<a href="#footnote-${index+1}" id="footnote-anchor-${index+1}"><sup>[${index+1}]</sup></a>`);

  $(`<li id="footnote-${index+1}">${footnoteText}</li>`)
  .append(`<a href="footnote-anchor-${index+1}"> [Retour]</a>`)
  .appendTo(footnoteDisplay);
```

Dans cet extrait de code nous réalisons deux opérations :

- Avec **append()**, nous ajoutons un élément <a> à notre élément
- Avec **appendTo()**, nous insérons notre élément <li> (et par extension, l'élément <a> associé) à notre élément

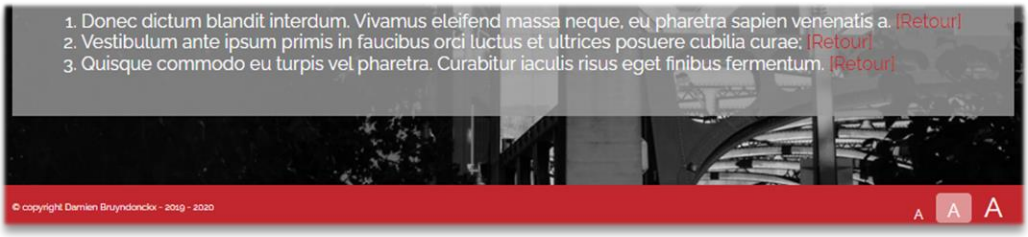
Le principe sera le même avec les fonctions « .prepend() » et « .prependTo() ». L'une va dans un sens et l'autre dans le sens contraire.

Utiliser la fonction wrap()

Documentation : [.wrap() | jQuery API Documentation]

Jusqu'à présent, nous avons créé notre liste <ol>, nos éléments <li> et <a>. A présent, nous allons utiliser la fonction « .appendTo() » pour injecter l'ensemble (l'élément <ol> donc) dans notre code HTML, comme vu précédemment.

```
18    footnoteDisplay.appendTo("main");
```

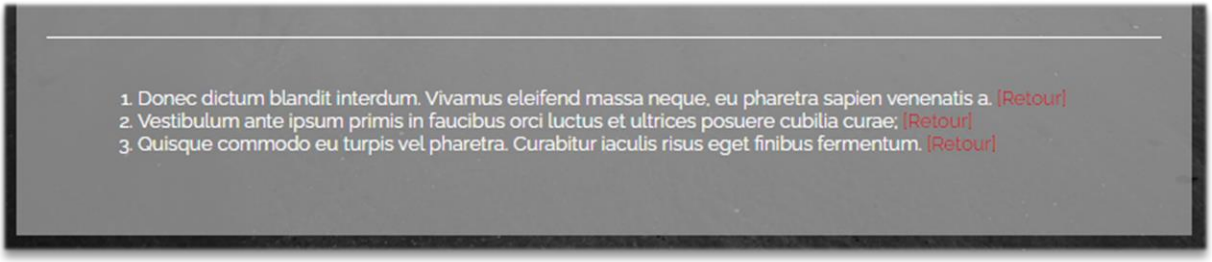


Nous allons à présent mettre en forme cette nouvelle liste...

La fonction « .wrap() » permet d'insérer un élément HTML autour d'un élément ciblé.

Ainsi, le code final ressemblera à cela :

```
$(document).ready(function () {
  let footnotes = $("span[data-footnote]");
  console.log(footnotes);
  if (footnotes) {
    let footnoteDisplay = $("<ol></ol>");
    footnotes.each(function (index, element) {
      let current = $(element);
      let footnoteText = current.data("footnote");
      current
        .css("text-decoration", "underline dashed")
        .after(
          `<a href="#footnote-${index + 1}" id="footnote-anchor-${index + 1}"><sup>[${index + 1}]</sup></a>`
        );
      $(`<li id="footnote-${index + 1}">${footnoteText}</li>`)
        .append(`<a href="#footnote-anchor-${index + 1}"> [Retour]</a>`)
        .appendTo(footnoteDisplay);
    });
    footnoteDisplay
    .appendTo("main")
    .wrap('<section id="footnotesDisplay"></section>');
  }
});
```



Chapitre 4 : Utiliser les évènements

Dans cette partie nous nous intéresserons aux évènements JavaScript. Nous travaillerons pour cela sur notre exemple précédent, mais un peu plus bas cette fois.

Ces 3 boutons « A » vont nous permettre de modifier la taille de la police pour permettre aux utilisateurs de choisir à leur convenance ce qui leur parait le plus agréable.

Le détail du code HTML est le suivant :

```
46      <footer id="mainFooter">
47          <p>&copy; copyright Damien Bruyndonckx - 2019 - 2020</p>
48          <p id="txtSize">
49          <button style="font-size: 12px;" data-size="0.8">A</button>
50          <button style="font-size: 16px;" data-size="1" class="active">A</button>
51          <button style="font-size: 24px;" data-size="1.5">A</button>
52      </p>
53      </footer>
```

Nous remarquerons l'utilisation de l'attribut « data-size » et de la classe « .active » sur l'un des bouton pour le garder en surbrillance afin d'indiquer le paramètre courant choisi.

Attacher des évènements

Il va falloir commencer par réfléchir aux éléments auxquels nous voulons attacher des évènements. En l'occurrence, il s'agit là des deux boutons « a » qui ne sont pas encore sélectionnés.

Pour cela, dans notre code HTML, nous allons devoir sélectionner les trois boutons contenus dans la balise de paragraphe **à l'exception** du bouton qui possède la classe « .active ».

Documentation : .on() | jQuery API Documentation

La fonction « .on() » va permettre d'attacher une écoute d'évènement à un élément.

Dans notre cas, nous aurons besoin de passer deux paramètres en arguments :

- Events : correspond à ou aux événements attendu(s)
- Handler : correspond à la fonction à exécuter à l'écoute de ces évènements

Notre script devrait ressembler à cela :

```
1   $(document).ready(function() {
2       $('#txtSize button')
3       .filter(':not(.active)')
4       .on('click', changeTxtSize);
5   })
```

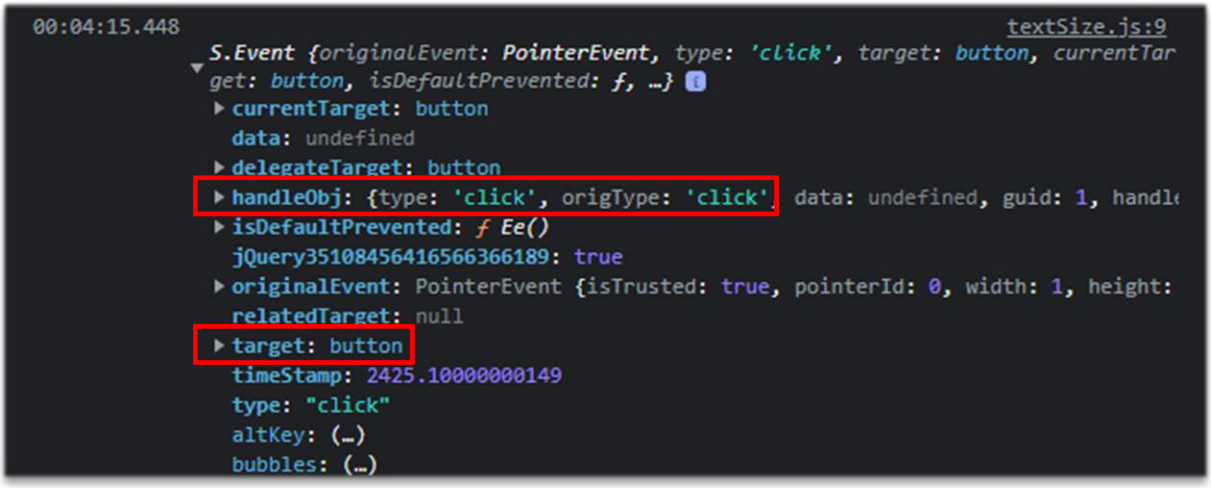
```
8   function changeTxtSize() {
9
10      // ? Ici commence notre fonction
11
12  }
```

Comprendre la propriété Target

Pour continuer, il va nous falloir trouver une solution pour déterminer la cible du clic. En effet, pour le moment, la fonction « changeTxtSize » se déclenche sans se soucier de notre cible.

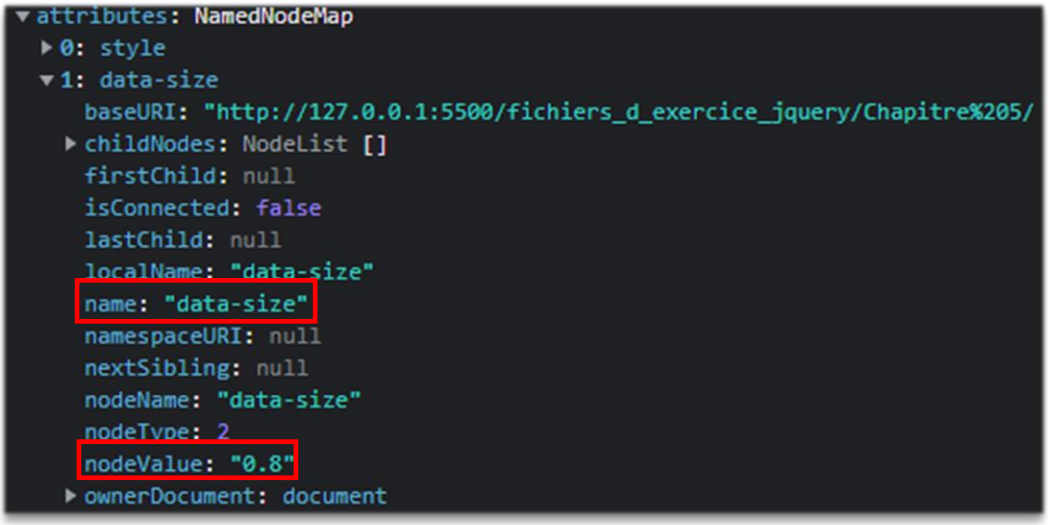
En JavaScript, un gestionnaire d'évènement (ici, notre fonction changeTxtSize) reçoit en argument l'évènement qui l'a déclenché.

```
8   function changeTxtSize(event) {
9       console.log(event);
10  }
```



On remarquera que maintenant, lorsque je clic sur le bouton, l'Object Target est bien relatif au bouton sur lequel je clic !

Si l'on fouille un peu dans les propriétés de cet éléments, nous retrouvons la valeur de l'attribut « data-size ».



En fin de compte, en utilisant les connaissances des chapitres précédents, notre fonction changeTxtSize() devrait ressembler à cela :

```
function changeTxtSize(event) {
    console.log(event);
    let clicked = $(event.target);
    let newSize = clicked.data('size');
    $('#pageContent').css('font-size', `${newSize}rem`);
}
```

Il va maintenant falloir manipuler la classe « .active » pour pouvoir rendre le tout fonctionnel.

Manipuler les classes CSS

Documentation : .addClass() | jQuery API Documentation

Documentation : .removeClass() | jQuery API Documentation

Grâce aux fonction « .addClass() » et « .removeClass() » nous allons pouvoir ajouter ou supprimer la classe « .active » de nos boutons :

```
function changeTxtSize(event) {
    console.log(event);
    let clicked = $(event.target);
    let newSize = clicked.data('size');
    $('#pageContent')
        .css('font-size', `${newSize}rem`);
    $('#txtSize button.active')
        .removeClass('active');
    clicked
        .addClass('active');
}
```

Un problème persiste pourtant. Impossible de revenir sur le bouton du milieu. Il va donc falloir détacher les évènements.

Détacher des évènements

Documentation : .off() | jQuery API Documentation

De la même façon que nous avons attaché une écoute d'évènement a l'aide de la fonction « .on() », nous allons pouvoir la détacher à l'aide de la fonction « .off() ». Ainsi, notre script devrait ressembler à cela :

```
function changeTxtSize(event) {
    console.log(event);
    let clicked = $(event.target);
    let newSize = clicked.data('size');
    $('#pageContent')
        .css('font-size', `${newSize}rem`);
    $('#txtSize button.active')
        .removeClass('active')
        .on('click', changeTxtSize);
    clicked
        .addClass('active')
        .off('click');;
}
```

Ici, nous attachons de nouveau un évènement au bouton qui perd sa classe « .active » et nous détachons l'écoute de celui qui la reçoit !
Et voilà, les boutons sont fonctionnels !

Utiliser les évènements raccourcis

Documentation : .click() | jQuery API Documentation

La fonction « .click() » produit... exactement le même résultat que la fonction « .on() » !

A cela près qu'il s'agit là d'une fonction « raccourcis ».En effet, avec cette fonction, le type d'évènement n'est plus à préciser, il est induit par le nom de la fonction.

Ainsi, mon bout de script précédent pourra tout à fait ressembler à cela :

```
$(document).ready(function() {
    $('#txtSize button')
    .filter(':not(.active)')
    .click(changeTxtSize);
})

function changeTxtSize(event) {
    console.log(event);
    let clicked = $(event.target);
    let newSize = clicked.data('size');
    $('#pageContent')
        .css('font-size', `${newSize}rem`);
    $('#txtSize button.active')
        .removeClass('active')
        .click(changeTxtSize);
    clicked
        .addClass('active')
        .off('click');;
}
```

Chapitre 5 : Animer la page web