jQuery

Michel Gagnon et Nikolay Radoev
École Polytechnique de Montréal

jQuery

- Bibliothèque Javascript pour simplifier la manipulation du DOM (entre autres)
  - Existe depuis 2006
  - Beaucoup de fonctionnalités désormais disponibles en HTML + JS natifs
  - Encore très utilisé par beaucoup de sites et dans beaucoup de ressources en ligne (ex: StackOverflow)
- Grand avantage: le code devient automatiquement adapté à tous les navigateurs
- Principe de base: des requêtes pour extraire les éléments qu'on désire manipuler
- Ajoute aussi certaines fonctionnalités qui n'existent pas dans le DOM (moins important en 2020)

La fonction **jQuery()** ou **$()**

- Elle est fondamentale: elle prend en argument (souvent un sélecteur CSS) et retourne un *objet jQuery* sur lequel un grand nombre de manipulations peuvent être réalisées
- Très souvent, lorsque la requête correspond à plusieurs éléments, cet objet est une collection d'items, qui se manipule comme un tableau
- On peut aussi exécuter une méthode sur cette collection d'items, ce qui aura pour effet de l'appliquer à chaque item de cette collection
- Exemples:
  - Pour changer la couleur de fond de tous les paragraphes, on pourrait écrire ceci (ne faites pas ça dans votre application!):

    $('p').css('background-color','#ccc')
  - Pour ajouter un gestionnaire de l'événement click à tous les éléments ayant la classe cliquerPourCacher:

    $('.cliquerPourCacher').click(function() { $(this).hide(); });

Comment invoquer la fonction **jQuery()**

1. On lui passe un sélecteur CSS
   - Si on lui passe un élément comme second argument, la recherche se fera seulement dans cet élément et ses descendants
     (ex. **$('p', document.getElementById('mondiv'))**)

2. On lui passe un élément, l'objet **document** ou l'objet **window**: dans ce cas, il retourne un objet jQuery qui englobe celui qui est passé en paramètre
   (ex. **$(document)**)

3. On lui passe du code HTML
   - On peut lui passer en second argument un objet qui indique les valeurs de certaines propriétés

On peut aussi lui passer comme second argument un élément qui servira de contexte (ex. **$('
', { 'class': 'unLien' })** ou **$('
')**)

1. On lui passe une fonction: celle-ci est exécutée lorsque la construction du DOM est terminée (on lui passe en argument le document)
   (ex. **$(function() { ... })**)

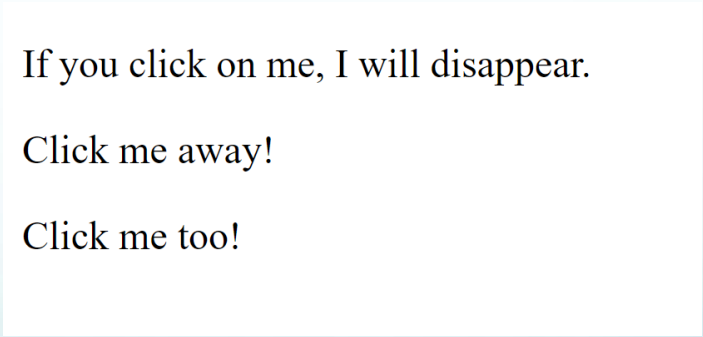
Exemple avec **$()**

```
<!DOCTYPE html>
<html>

<head></head>

<body>
    <p>If you click on me, I will disappear.</p>
    <p>Click me away!</p>
    <p>Click me too!</p>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></s
cript>
    <script>
        $(document).ready(function () {
            $("p").click(function () {
                $(this).hide();
            });
        });
    </script>
</body>

</html>
```



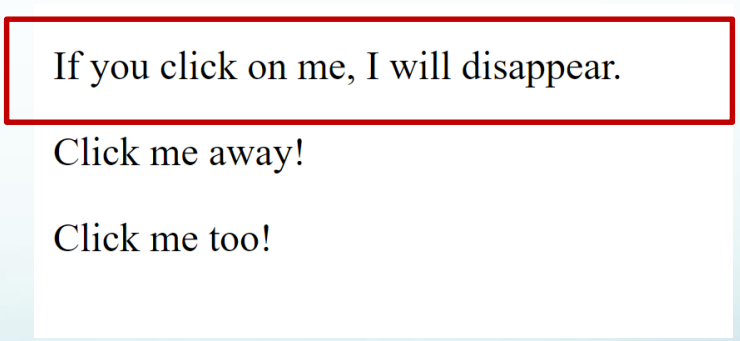
Exemple avec **$()**

```
<!DOCTYPE html>
<html>

<head></head>

<body>
    <p>If you click on me, I will disappear.</p>
    <p>Click me away!</p>
    <p>Click me too!</p>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></s
cript>
    <script>
        $(document).ready(function () {
            $("p").click(function () {
                $(this).hide();
            });
        });
    </script>
</body>

</html>
```



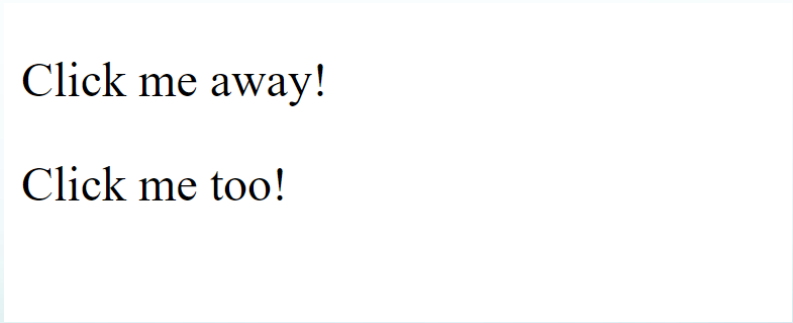
Exemple avec **$()**

```
<!DOCTYPE html>
<html>

<head></head>

<body>
    <p>If you click on me, I will disappear.</p>
    <p>Click me away!</p>
    <p>Click me too!</p>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></s
cript>
    <script>
        $(document).ready(function () {
            $("p").click(function () {
                $(this).hide();
            });
        });
    </script>
</body>

</html>
```



Comment itérer sur une collection d'items

- Méthode **each()**: reçoit en paramètre une fonction qui sera appelée pour chaque item de la collection
  - La fonction reçoit l'indice de l'item comme premier argument et l'élément concerné comme second argument
  - **this** représente aussi l'élément concerné
  - Si la fonction retourne **false**, l'itération s'arrête
- Itération implicite: beaucoup de méthodes itèrent de manière implicite sur tous les items:
  - **$('button').click(function() { … })**

Méthode **map()**

- Comme **each()**, elle reçoit en paramètre une fonction qui sera appelée sur tous les items de la collection
- Retourne une nouvelle collection qui contient tous les résultats obtenus par l'application de la fonction à chaque item
- ES5 ajoute les méthodes **map()** et **filter()** sur l'objet Array, fortement inspirés de jQuery
- Exemple:
  - **$('div').map(function() {**
    return this.id;
    }).toArray().sort()

Méthode **index()**

- Si elle reçoit un élément en paramètre, elle retourne l'indice de cet élément
- Si elle reçoit un sélecteur CSS, elle retourne l'indice du premier élément qui correspond à ce sélecteur
- Exemple:

```
let listItem = document.getElementById( ‘‘myId" );
let itemAtString = "Index: " + $( "li" ).index( listItem );
```

Méthode **is()**

- Prend un sélecteur CSS comme argument
- Retourne **true** si l'élément correspond au sélecteur en paramètre
- Exemple:

```
$('div').each(function() {
  if ($(this).is(':hidden')) return;
});
```

Manipulation des attributs

- La même méthode est utilisée pour lire la valeur courante et pour fixer une nouvelle valeur
- Lorsqu'il s'agit d'une collection, l'opération est appliquée à tous les items de la collection, s'il s'agit de fixer une nouvelle valeur
- S'il s'agit de lire la valeur courante, cela sera appliqué seulement sur le premier item
- On peut passer comme argument un objet pour fixer la valeur de plusieurs attributs
- Souvent, on peut aussi passer une fonction qui retournera la nouvelle valeur désirée: dans ce cas, elle sera appliquée à l'item et recevra deux arguments (l'indice de l'item et sa valeur courante)

Méthodes de manipulation des attributs

- attr()
- css()
- addClass(), removeClass(), toggleClass()
- val()
- text() et html()
- width(), height(), innerWidth(), outerWidth(), ...
- data(), removeData()

Modification de la structure du document

- Ajout d'éléments:
  - append(), prepend(), before(), after(), replaceWith()
  - appendTo(), prependTo(), insertBefore(), insertAfter(), replaceAll()
- Copie d'éléments:
  - clone()
  - Par défaut, ne copie pas les gestionnaires d'événements attachés
- Enrobage: wrap(), wrapInner(), wrapAll()
- Retrait d'éléments: empty(), remove(), detach(), unwrap()

Événements en jQuery

- Pour chaque événement du DOM, il y a une méthode (par exemple: click = click())
- Ajouts:

  hover(): prend deux fonctions en argument, soit une pour **mouseenter** et une autre pour **mouseleave**

  toggle(): pour chaque clic, retire ou affiche l'item en alternance

  toggleClass(): pour chaque clic, retire ou ajoute la classe en alternance
- Le gestionnaire reçoit en paramètre un objet correspondant à l'événement (attributs de l'objet: target, relatedTarget, which, type, pageX, pageY etc. ; méthodes: preventDefault(), stopPropagation()
- Si un gestionnaire retourne **false**, la propagation et le comportement par défaut sont éliminés

Événements « vivant »

- Supposons que le code suivant s'exécute:
  $('a').click(handlerFunction)
- Si d'autres éléments **<a>** s'ajoutent par la suite, le gestionnaire (*handlerFunction*) n'y sera pas attaché
- Pour obtenir ce comportement, il faut utiliser la méthode **delegate()**:
  $(document).delegate('a','click',handlerFunction)
- Principe: lorsque la propagation atteint l'élément sur lequel **delegate()** est appelé (le document, dans l'exemple précédent), on détermine si la cible correspond au sélecteur et si c'est le cas on exécute le gestionnaire sur cette cible

Méthodes de sélection

- On peut utiliser les sélecteurs CSS comme dans une règle CSS normale
- Ajout de pseudo-classes:
  :animated, :button, :checkbox, :contains(text), :eq(n), :even, :file, :first (ne pas confondre avec :first-child) :gt(n), :has(sel), :header, :hidden, :image, :input, :last, :lt(n), :odd, :parent, :password, :radio, :reset, :selected :submit, :text, :visible
- Exemple: $('**p:nth-child(3):not(:has(a))**')

Méthodes de sélection

- On peut aussi utiliser des méthodes de sélection définies par jQuery
- Méthode **slice(m,n)**: sélectionne les items des indices m à n
- Méthode **filter()**: sélectionne les items correspondant au critère passé en paramètre:
  - Un sélecteur CSS
  - Une autre collection jQuery
  - Une fonction prédicat (qui retourne une valeur booléenne) (ex. **$('p').filter('.middle')**)
- Méthode **not()** (ex. **$('p').not(':even')**)
- Méthode **has()** (ex. **$('p').has('strong')**)

Méthodes de sélection

- Méthode **find()**: cherche parmi les descendants de l'item concerné

  Exemple: **$('div').find('a')** pour obtenir tous les éléments <a> inclus dans un

- Méthodes **children()**, **contents()**, **next()**, **prev()**, **nextAll()**, **prevAll()**, **siblings()**, **parents()**

Méthodes de sélection

- Méthode **end()**: pour revenir à niveau précédent dans la chaîne (contrairement au CSS qui ne peut que descendre l'arborescence)

- Exemple:

```
let divs = $('div');
let paragraphs = divs.find('p');
paragraphs.addClass('important');
divs.css('border','solid black 1pt')
```

est équivalent à ceci:

$('div').find('p').addClass('important').end().css(...)

ou encore:

$('div').css(...).find('p').addClass('important')