1 - Contexte

1.1 Librairie JS

Une librairie est un ensemble de fonctionnalités clés en main fournies aux développeurs. Son rôle est de faciliter le développement grâce à des outils mis à disposition des développeurs.

En règle générale, une librairie est spécialisée dans un domaine précis. Cela peut être par exemple la gestion de formulaires, la gestion de la géolocalisation, la data visualisation, des animations visuelles ou encore des librairies permettant de créer des interfaces utilisateurs comme React.

1.2 Naissance de Jquery

En 2006, Microsoft domine le marché des navigateurs web grâce à Internet Explorer avec plus de 80% de marché. Toutefois, Internet Explorer n'interprète pas les dernières versions de Javascript de la même manière que des navigateurs plus récents (Firefox notamment). Les développeurs testent souvent leur développement sous Firefox puis l'adapte pour être compatible avec Internet Explorer.

C'est dans ce contexte que la librairie Jquery voit le jour. Elle propose de s'affranchir des différences entre les navigateurs web en proposant des fonctionnalités clés en main qui intègrent les particularités de chaque navigateur, notamment Internet Explorer.

1.3 Avantage de Jquery

Jquery ne va pas se contenter de proposer une gestion simplifiée des navigateurs, cette librairie va également offrir de nombreuses fonctionnalités pour simplifier l'utilisation de Javascript à travers la mise à disposition d'outils adaptés aux développeurs web.

Jquery propose de :

S'affranchir des différences entre les navigateurs web
Proposer des outils simplifiés pour manipuler le DOM et gérer les évènements en raccourcissant certaines syntaxes notamment
Offrir des fonctionnalités pour ajouter des effets visuels et animations

D’autres librairies proposaient à cette époque des fonctionnalités similaires mais Jquery va alors très vite être adoptée par la communauté grâce à sa syntaxe très intuitive et se développer pour être utilisée sur une grande majorité de sites web.

1.4 Utilisation dans les entreprises

Au fil des années, la librairie n’a cessé de se développer et d’être adoptée par de plus en plus de sites web. Le développement de sites web utilisant la technologie Ajax a encore multiplié l’utilisation de Jquery.

Actuellement, Jquery est utilisé sur plus de 80% des 100 000 sites web ayant le plus d’audience.

Il s’agit d’ailleurs de la librairie Javascript la plus utilisée depuis plus de 12 ans.

1.5 Déclin

Malgré ce succès, Jquery souffre de plusieurs critiques :

> Sa lourdeur : effectivement, pour utiliser une librairie JS, il va falloir appeler le fichier contenant les différentes fonctionnalités mises à disposition. Au fil du temps, de plus en plus de fonctionnalités ont vu le jour provoquant une hausse du poids du fichier.

> Plus le poids est important, plus cela ralentit le temps de chargement de nos pages web.
>
> Jquery s’est adapté en proposant rapidement des versions minifiées (les espaces et retours chariots sont retirés pour gagner de nombreux octets et alléger le fichier de base) puis en proposant des versions allégées (slim), sans ajax et sans effets.

Moins utiles depuis les dernières versions de Javascript : chaque nouvelle version de Javascript apporte son lot de fonctionnalités qui, peu à peu, permettent de couvrir les besoins qui étaient auparavant satisfaits par Jquery
Dépassé par des librairies modernes : Certaines librairies concurrencent sur certains aspects Jquery

C'est pourquoi, au fur et à mesure, des évolutions ou refonte de sites, Jquery est peu à peu mis de côté.

1.6 Faut-il apprendre Jquery ?

Comme vu ci-dessus, Jquery est encore incontournable sur de nombreux sites web. Tous les développeurs web doivent donc maîtriser cette librairie. Il est quasiment certain que vous soyez amené à l'utiliser dans votre carrière.

Jquery est évidemment très utilisée sur des projets web ayant une certaine ancienneté. Si vous travaillez sur un site ayant quelques années, il est possible de retrouver l'utilisation de certaines fonctionnalités utilisant la librairie Jquery.

Sur les nouveaux projets ou sur des projets de refonte de site web, les librairies JS moderne telles que React sont davantage adoptées. L'émergence de ces librairies pousse Jquery vers la sortie.

Toutefois, quand on compare l'utilisation actuelle de Jquery (80%) à celle de React (5%) on se rend compte que la disparition de Jquery va être très lente et mettre de nombreuses années.

1.8 C'est quoi le Vanilla JS

Avec le développement de Jquery, beaucoup de développeurs ont appris le Javascript à travers l'utilisation de la librairie Jquery. À tel point qu'on en a fini par oublier les fondamentaux de Javascript.

Il est risqué de ne maîtriser un langage qu'à travers une librairie, de ne pas connaître les subtilités du langage et d'avoir des difficultés à passer outre cette librairie.

C’est pourquoi, malgré la connaissance d’une librairie comme Jquery, il est primordial de maîtriser le code de base de Javascript (sans librairie) qu’on appelle Vanilla JS.

Vanilla JS est d’une certaine sorte le nom commercial du Javascript de base permettant de vanter les qualités de l’utilisation du Javascript “nu” en lieu et place des librairies Javascript.

2 - Préparation de l’environnement

Pour utiliser la librairie Jquery, il est nécessaire d’inclure le fichier contenant l’ensemble des fonctionnalités de Jquery.

Jquery, comme de nombreuses librairies, met à disposition ce fichier sur un CDN (content delivery network).

En plus d’une garantie de délivrabilité de notre fichier, ce réseau de diffusion de contenu permet, grâce à son ensemble de serveurs reliés en réseau à travers le monde, de proposer le contenu situé sur le serveur le plus proche et d’optimiser ainsi les temps de chargement du fichier.

Autre avantage à l’utilisation d’un CDN, lorsqu’un utilisateur arrive sur votre site, il y a de fortes chances qu’il soit déjà passé par un autre site utilisant ce même fichier. Ce fichier sera alors présent dans le cache du navigateur et n’aura pas besoin d’être chargé. Votre site se chargera alors plus rapidement.

Jquery propose plusieurs versions de son fichier :

uncompressed

> Cette version est la version de base de Jquery avec l’ensemble des fonctionnalités. Elle n’est pas optimisée en terme de poids et est donc destinée à votre environnement de développement, si vous souhaitez pouvoir lire le fichier et comprendre les différentes fonctions développées

compressed (minify)

Cette version est identique à la précédente. Seule exception, elle est compressée : les espaces et les retours à la ligne ont été supprimés pour optimiser son poids et permettre un chargement plus rapide. Cette version est donc destinée à votre site en production. Elle ne permettra pas de lire facilement le contenu du fichier car tout le code se retrouve sur une même ligne.

uncompressed slim

C’est une version de développement allégée. Elle ne contient pas l’ensemble des fonctionnalités du script d’origine. Les parties Ajax et Effets ont été retirées pour gagner en poids et donc en temps de chargement. Si vous n’utilisez pas ces fonctionnalités là, il est alors conseillé d’utiliser la version slim, moins longue à se charger.

compressed slim

C’est la version slim optimisée pour votre environnement de production. Les données ont été compressées.

Pour inclure le fichier, il est conseillé de le placer dans la partie <head> de votre fichier HTML.

```html
<!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>Jquery</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>


    <h1 id="todo">Todo List</h1>

    <p>RDV chez le dentiste lundi 18h</p>

    <p>Cours de tennis mercredi 14h30</p>




    <script src="script.js"></script>
</body>
</html>
```

Lien du

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

3 - Coder en Jquery

3.1 1ere ligne de code

Maintenant que le fichier Jquery est appelé, il va être possible d'utiliser Jquery dans nos fichiers Javascript

Fichier HTML

```html
<!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>Jquery</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>

    <h1 id="todo">Todo List</h1>

    <p>RDV chez le dentiste lundi 18h</p>

    <p>Cours de tennis mercredi 14h30</p>

    <script src="script.js"></script>
</body>
</html>
```

Fichier JS

```javascript
$('#todo-1').text("RDV chez le dentiste mardi 9h");
```

Ici, dans notre fichier Javascript, nous pouvons cibler l'élément d'identifiant "todo-1" grâce à Jquery : $('#todo-1').

Puis nous pouvons utiliser la méthode text pour modifier le contenu texte de notre élément.

Le contenu du premier paragraphe va alors changer dynamiquement pour prendre la nouvelle valeur fournie : "RDV chez le dentiste mardi 9h".

3.2 Mixer Jquery et Vanilla JS

Jquery est une librairie qui apporte une écriture simplifiée et de nouvelles fonctionnalités clés en main. Toutefois, il est tout à fait possible de mixer Jquery avec du Vanilla JS.

```
var txt = "RDV chez le dentiste mardi 9h";
txt = txt.toUpperCase();
$('#todo-1').text(txt);
```

Dans cet exemple, nous allons déclarer une variable txt contenant du texte.

Nous utilisons Vanilla JS pour mettre ce contenu en majuscule grâce à la méthode toUpperCase().

Puis nous utilisons Jquery pour cibler l'élément d'identifiant "todo-1" et lui affecter la valeur créée précédemment.

Attention : Même si cela est possible, on va éviter d'utiliser dans un même script les mêmes fonctionnalités Jquery et Vanilla JS.

```html
<!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>Jquery</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>


    <h1>Todo List</h1>

    <p id="todo-1">RDV chez le dentiste lundi 18h</p>

    <p id="todo-2">Cours de tennis mercredi 14h30</p>




    <script src="script.js"></script>
</body>
</html>
```

```
$('#todo-1').text("RDV chez le dentiste mardi 9h");
document.getElementById('todo-2').textContent = "RDV chez le dentiste mercredi 9h";
```

Pour garder une cohérence sur son site, il est effectivement conseillé de faire un choix sur la librairie à utiliser.

```
$('#todo-1').text("RDV chez le dentiste mardi 9h");
$('#todo-2').text("RDV chez le dentiste mercredi 9h");
```

Ici, nous décidons d'utiliser Jquery pour cibler tous nos éléments.

Les évènements (jQuery)

1 - Contexte

1.1 C’est quoi un événement ?

C’est un signal invisible souvent lié à une action de l’utilisateur dans le navigateur. Celui-ci va être à l’écoute de tous les signaux qui se produisent sur la page Web et va transmettre ces informations à nos scripts via le gestionnaire d'événements.

Javascript est ce qu’on appelle un langage événementiel, le code déclaré dans nos fichiers Javascript ne va pas forcément s'exécuter de manière séquentielle (ligne par ligne) mais souvent attendre qu’un événement soit détecté pour s'exécuter.

1.2 Exemple d'événements

Les événements sont souvent liés à une action de l’utilisateur :

- Un clic sur un bouton
- Passer sa souris sur un élément HTML
- Scroller la page
- Une touche du clavier saisie
- Le changement de valeur d’un élément HTML (input, textarea, …)

Tous ces événements émettent des signaux qui seront captés par le navigateur.

1.3 Pour quoi faire ?

Associée au DOM, la gestion des événements va prendre toute son importance.

Rappelez-vous des possibilités offertes par la gestion dynamique du DOM.

Par exemple, pour Gmail, il était possible de modifier dynamiquement certains blocs de la page pour permettre d’afficher des informations comme le fait qu’un mail est lu.

À quel moment cette modification doit-elle être faite ? Lorsque l'utilisateur a cliqué sur le mail en question. C'est bien cette action qui va déclencher une modification du contenu de notre page.

C'est là qu'interviennent les événements ! C'est bien le signal du clic sur le mail qui doit déclencher une modification. C'est donc grâce aux événements qu'on pourra détecter le clic de l'utilisateur et effectuer cette action et modifier le DOM.

2 - Capter les événements

2.1 Définir l'élément que vous voulez écouter

Pour capter le signal émis, la première étape consiste à cibler l'élément HTML qu'on souhaite écouter, notamment par son identifiant :

```
$('#todo-1')
```

Jquery propose la fonction sélecteur $() pour cibler un élément.

Comme en CSS, lorsque nous ciblons un élément par son identifiant, nous précédons l'identifiant d'un #.

Ici, nous ciblons un élément portant l'identifiant "todo-1" : #todo-1

2.2 Les 2 étapes à créer

Deux étapes sont ensuite nécessaires pour gérer ces événements :

- Mise sur écoute

Cette étape consiste à mettre sur un écoute un ou plusieurs éléments HTML en précisant le type d'élément qu'on souhaite écouter (un clic, le passage de la souris, le scroll, …)

```
$('#todo-').click(function(){
    console.log('Au moment du click');})
```

Maintenant que la mise sur écoute est en place, nous allons pouvoir fournir des informations sur l'action à effectuer lorsque cet événement va être déclenché.

Dans notre exemple avec Gmail, nous allons mettre sur écoute le clic sur le libellé d'un mail.

Action à effectuer

C'est l'action à effectuer, le traitement à faire, l'action à réaliser.

Dans notre exemple, il s'agirait de modifier la police du libellé (de gras à normal) pour nous indiquer que le message a bien été lu.

```
$('#todo-').click(function(){

    $(this).css("font-size","regular")

})
```

2.3 Les événements avec Jquery

Fichier .html

```html
<!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>Jquery</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>


    <h1>Todo List</h1>

    <p id="todo-1">RDV chez le dentiste lundi 18h</p>

    <p id="todo-2">Cours de tennis mercredi 14h30</p>




    <script src="script.js"></script>
</body>
</html>
```

Fichier .js

```
$('#todo-1').click(
  function(){
    console.log("click détecté !");
  }
);
```

Le gestionnaire d'événement va se charger de surveiller les signaux pour déclencher un bloc de code, une action. C'est lui qui a la responsabilité de retrouver le bloc de code associé à cette surveillance pour le déclencher le moment opportun.

Jquery propose un ensemble de méthodes pour gérer les différents évènements : https://api.jquery.com/category/events

Ici, nous utilisons la méthode "click" qui permet de mettre sur écoute le clic sur l'élément souhaité.

Cette méthode contient un premier argument, la fonction de callback, qui sera appelée dès que l'événement est détecté. ici, nous ne déclencherons qu'un console.log()

Que se passe-t- il à l'exécution du code ?

Au chargement de la page, une écoute est mise en place sur le clic de l'élément portant l'identifiant "todo-1". Au clic sur cet élément, un signal est émis. Le gestionnaire d'événement capte le signal. La méthode click exécute alors l'action fournie en second argument. La fonction est alors exécutée et "clic détecté !" s'affiche dans la console.

3 - L'asynchrone

La mécanique de la gestion des événements nous amène à devoir comprendre la notion de synchrone / asynchrone.

Comme dans une recette de cuisine, plusieurs instructions doivent s'exécuter étape par étape. Tant que la 1ère étape n'est pas terminée, l'autre ne commence pas. On parle alors d'exécution séquentielle du code. Les étapes s'exécutent de manière synchrone.

Toutefois, dans une recette de cuisine, il se peut qu'on mutualise certaines étapes pour gagner du temps et que certaines étapes dépendent alors de la fin de l'exécution d'une autre étape.

> Mettre l'eau à bouillir : quand l'eau bout, mettre les pâtes à cuire
> Faire cuire la sauce

Hacher les oignons

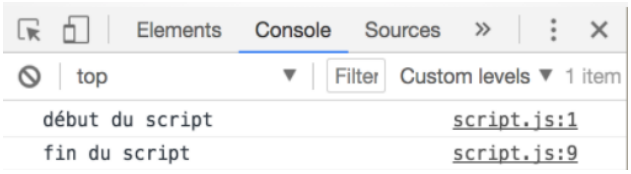
Ici, nous allons exécuter nos étapes de manière séquentielle. Toutefois, nous n'allons pas attendre que l'eau bout pour avancer dans nos autres tâches.

Nous allons avancer dans nos différentes tâches et attendre que le signal "l'eau bout" nous parvienne pour déclencher l'action de mettre les pâtes à cuire.

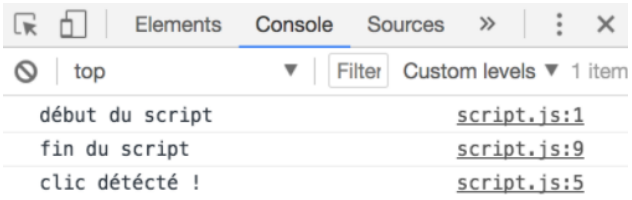
On parle alors d'exécution asynchrone de cette tâche. Cette tâche est mise de côté et s'exécute uniquement lors de la remontée d'un signal.

```
console.log("début du script");


$('#todo-1').click(
  function(){
    console.log("clic détecté !");
  }
);


console.log("fin du script");
```

Dans cet exemple, la fonction de callback ne va se lancer qu'au clic sur l'élément todo-1. On parle d'exécution asynchrone



Voilà pourquoi, le console.log() "clic détecté" ne va pas s'afficher au chargement de la page.



Au clic sur l'élément, le console.log() “clic détecté” s’affiche alors.

```
console.log("début du script");


$('#todo-1').click(
  function(){
    console.log("début de la fonction de callback");
    console.log("click détecté !");
    console.log("fin de la fonction de callback");
  }
);


console.log("fin du script");
```

La fonction de callback peut contenir plusieurs instructions. Ici, plusieurs console.log().

À l’intérieur de notre fonction de callback, l'exécution est bien synchrone. Quand le clic est détecté, la fonction est appelée et les différentes instructions vont s'exécuter les unes après les autres au sein de cette fonction de callback.

4 - Écouter plusieurs éléments

Imaginons que nous souhaitions mettre sur écoute plusieurs éléments

```html
<!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>Jquery</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>


    <h1>Todo List</h1>

    <p id="todo-1">RDV chez le dentiste lundi 18h</p>

    <p id="todo-2">Cours de tennis mercredi 14h30</p>




    <script src="script.js"></script>
</body>
</html>
```

Dans cet exemple, si nous souhaitons écouter les deux paragraphes “todo-1” et “todo-2”, nous pouvons mettre deux écoutes en place, une sur chaque paragraphe

```
$('#todo-2').click(
  function(){
    console.log("click sur todo-2 détecté !");
  }
);

$('#todo-1').click(
  function(){
    console.log("click sur todo-1 détecté !");
  }
);
```

Chaque bloc gère un signal différent :

Une première écoute sur l’élément “todo-1” qui affichera au clic “click sur todo-1 détecté”.

Une deuxième écoute sur l’élément “todo-2” qui affichera au clic “click sur todo-2 détecté”.

Cela va fonctionner mais on imagine les limites :

> Si nous avons 100 paragraphes à écouter, il faudra créer 100 blocs
> À chaque modification souhaitée sur l’action à faire, il faudra mettre en place cette modification dans chaque bloc

Plutôt que de cibler chaque élément par son identifiant, nous allons cibler un ensemble d'éléments grâce aux balises HTML ou bien grâce aux classes de chaque élément afin de factoriser le traitement.

Jquery nous propose une utilisation extrêmement simplifiée de ces écoutes : plus besoin de boucler sur l'ensemble des éléments d'un paragraphe pour poser une écoute sur chacun.

```
$('p').click(
    function(){
        console.log("click sur un paragraphe détecté !");
    }
);
```

Ici, notre objectif est de cibler l'ensemble des éléments paragraphe, de les mettre sur écoute et, au clic sur un paragraphe, d'effectuer une même action : le console.log()

Avec Jquery, c'est très simple, il suffit de cibler les balises <p> grâce à $('p').

De mettre sur écoute le clic sur tous ces éléments : $('p').click();

Puis de définir la fonction de callback en premier argument de la méthode clic .

Chaque paragraphe est désormais mis sur écoute.

Si vous souhaitez cibler uniquement certains paragraphes, il sera judicieux d'utiliser les classes pour cibler uniquement certains éléments :

```html
<!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>Jquery</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>

    <h1>Todo List</h1>

    <p id="todo-1" class="todo">RDV chez le dentiste lundi 18h</p>

    <p id="todo-2" class="todo">Cours de tennis mercredi 14h30</p>

    <p id="todo-3">Rdv Annulé</p>




    <script src="script.js"></script>
</body>
</html>
```

ci, nous souhaitons mettre sur écoute les éléments “todo-1” et “todo-2” mais pas l’élément “todo-3”.

Nous avons alors ajouté une classe “todo” sur nos deux premiers éléments.

```
$('.todo').click(
   function(){
      console.log("click sur un paragraphe détecté !");
   }
);
```

Nous pouvons cibler les éléments de la classe “todo” en précédant le nom de la classe par un point, de la même manière qu’en CSS.

5 - Connaître l'élément déclencheur : $(this)

```html
<!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>Jquery</title>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>

    <h1>Todo List</h1>

    <p id="todo-1" class="todo">RDV chez le dentiste lundi 18h</p>

    <p id="todo-2" class="todo">Cours de tennis mercredi 14h30</p>

    <p id="todo-3">Rdv Annulé</p>




    <script src="script.js"></script>
</body>
</html>
```

```
$('p').click(
   function(){
     console.log("click sur un paragraphe détecté !");
   }
```

```
);
```

En factorisant le traitement, nous avons perdu une information essentielle : celle de connaître quel élément a été cliqué.

Effectivement, désormais, lorsqu'on clique sur un paragraphe, l'information s'affiche dans la console mais nous ne sommes pas capables de connaître l'élément sélectionné.

On sait qu'un paragraphe a été cliqué, mais on ne sait pas lequel.

Comme avec Javascript, Jquery propose un moyen de l'identifier via un "mot magique" sensiblement similaire au niveau de la syntaxe $(this)

```
$('p').click(
    function(){
        console.log( $(this) );
    }
);
```

Ici, dans notre exemple, nous affichons dans la console l'élément paragraphe du DOM sur lequel on a cliqué.

Évidemment, nous allons pouvoir manipuler cet élément pour pouvoir par exemple modifier son contenu, récupérer différentes informations de cet élément ou récupérer le texte de celui-ci :

```
console.log($(this).text());
```

6 - Événement & DOM

Grâce au mot magique $(this), nous pouvons désormais manipuler l'élément HTML ciblé.

$(this) vous permet de récupérer des informations sur cet élément, modifier certaines informations, masquer l'élément par exemple.

```
$('p').click(
    function(){
        console.log($(this).text());
    }
);
```

Ici, on met sur écoute l'ensemble des éléments paragraphe. Au clic sur un des éléments, nous allons récupérer le texte de cet élément que nous allons afficher dans la console.

```
$('p').click(
    function(){
        $('#todo-1').text("RDV chez le dentiste mardi 9h");
    }
);
```

Ici, on met sur écoute l'ensemble des éléments paragraphe. Au clic sur un des éléments, nous allons modifier son texte pour le remplacer par "rendez vous annulé". Jquery propose de fournir un argument à la méthode text() pour modifier directement le texte de l'élément.

À chaque fois qu'un clic a lieu sur un paragraphe, son texte est remplacé

```
$('p').click(
    function(){
        $('#todo-1').remove();
    }
);
```

Ici, on met sur écoute l'ensemble des éléments paragraphe. Au clic sur un des éléments, nous allons supprimer l'élément.

Comme avec Javascript, il est possible de calculer le nombre d'éléments :

```
console.log($('p').length);
```

Pour parcourir chacun des paragraphes, Jquery propose la méthode each()

```
$("p").each(
    function() {
        console.log( $(this).text() );
    }
);
```

On cible tous les paragraphes. On boucle sur chacun d'entre eux et pour chacun, on exécute la fonction de callback fournie en premier argument :

Dans cette fonction de callback, on utilise le mot $(this) pour cibler l'élément en question et afficher son contenu texte.

Cette boucle va donc parcourir l’ensemble des éléments paragraphe et, pour chacun, afficher son texte dans la console du navigateur.