jQuery

Roi Yehoshua
2018

What is jQuery?

- jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML
- Originally created in January 2006 by John Resig
- As of June 2018, jQuery is used on 73% of the top 1 million websites, and by 22.4% of all websites
- jQuery main features are:
  - DOM element selections
  - DOM traversal and modifications
  - Events
  - CSS manipulation
  - Effects and animations
  - Ajax
  - Cross-browser support
  - Extensibility through plug-ins

Write Less, Do More

- Performing simple tasks using raw JavaScript can result in dozens of lines of code
- For example, to discover which radio element of a radio group is currently checked and to obtain its value attribute, we have to write the following code:

```
let checkedValue;
let elements = document.getElementsByName("some-group");
for (let i = 0; elements.length; i++) {
    if (elements[i].checked) {
        checkedValue = elements[i].value;
        break;
    }
}
```

- Contrast that with how it can be done using jQuery:

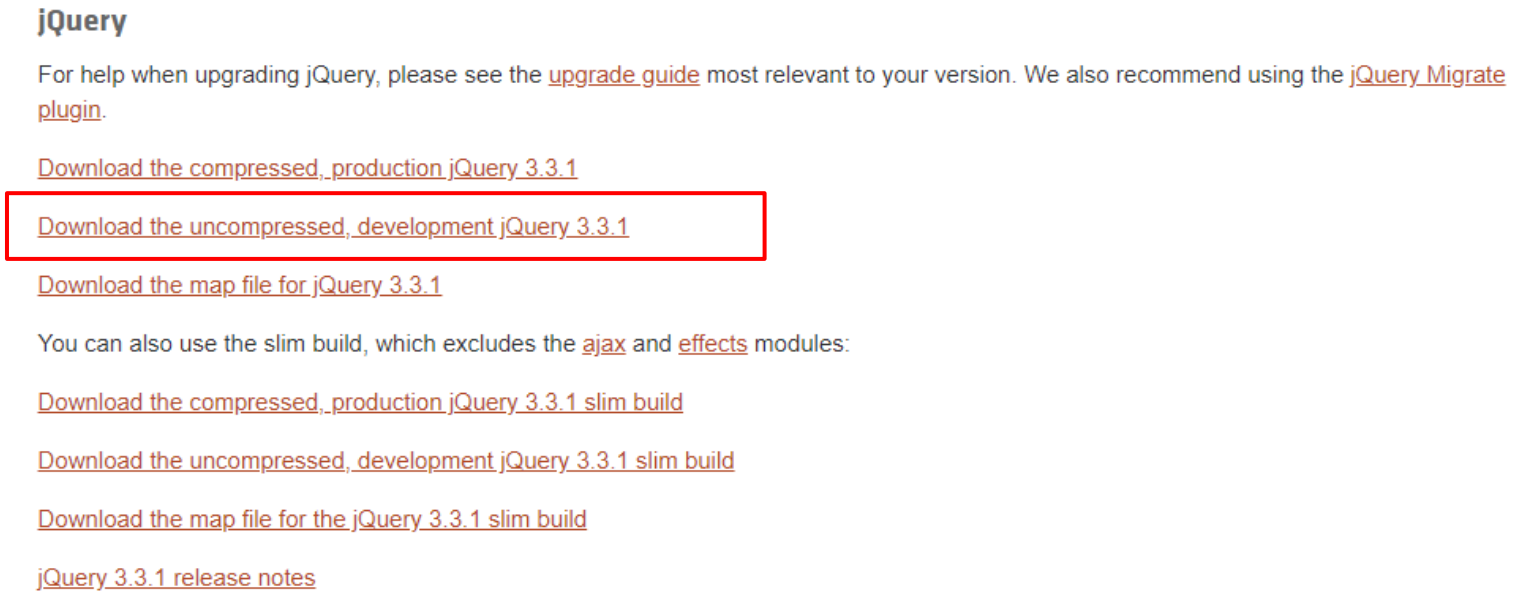
```
let checkedValue2 = $("input:radio[name='some-group']:checked").val();
```

Adding jQuery to Web Pages

- There are several ways to start using jQuery on your web site
- You can:
  - Download the jQuery library from jQuery.com
  - Include jQuery from a CDN, like Google
- There are two versions of jQuery available for downloading:
  - Production version - this is for your live website because it has been minified and compressed
  - Development version - this is for testing and development (uncompressed and readable code)
- Both versions can be downloaded from jQuery.com
- The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag

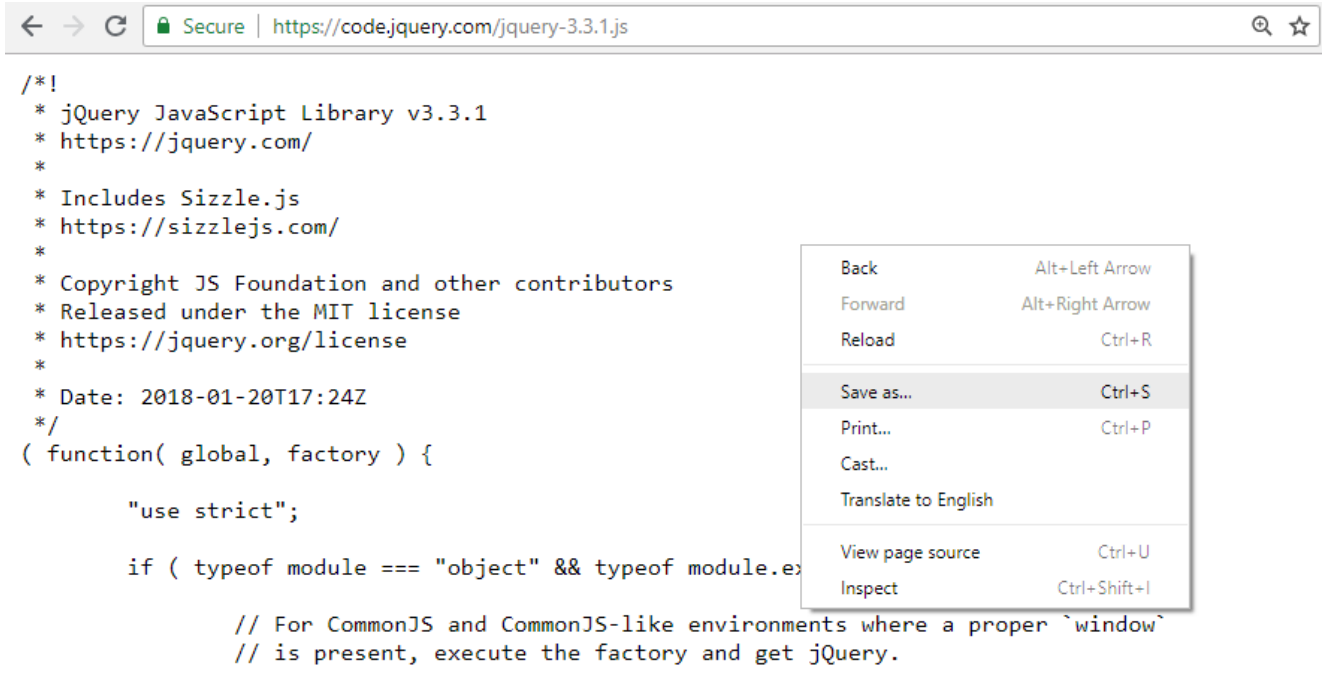
Downloading jQuery

- Let's download jQuery's latest development version (3.3.1 as of August 2018)
- Go to http://jquery.com/download/
- Choose Download the uncompressed, development jQuery 3.3.1



Downloading jQuery

- If you click the link, the jQuery script will open inside the browser
- Right-click the page and choose Save as… to save the script to your machine
- Place the script in the same directory as the pages where you wish to use it



Downloading jQuery

- Create a new HTML page named jQueryDemo.html, place it in the same directory as the jQuery script, and add the following script tag to it:

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>jQuery Demo</title>
</head>
<body>
    <script src="jquery-3.3.1.js"></script>
</body>
</html>
```

- Now you are ready to use jQuery in your own script tags (that appear after the jQuery script tag)

jQuery CDN

- If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network)
- There are several servers hosting jQuery, e.g., Google and Microsoft
- For example, to use jQuery from Google type:

```
<body>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</body>
```

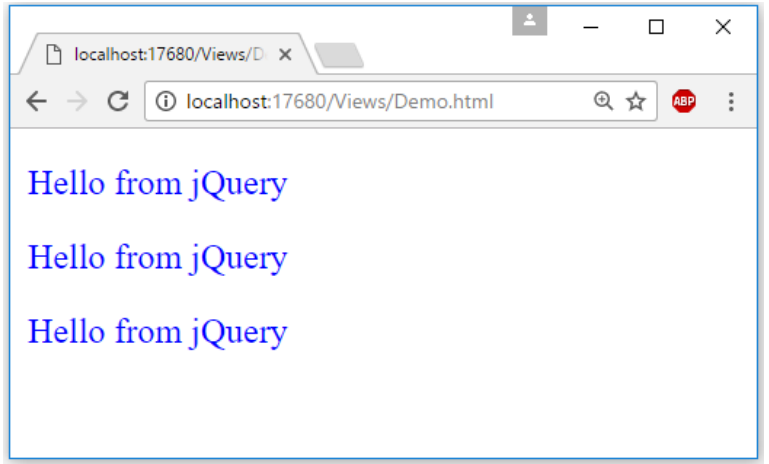
- Advantage of using CDN in production is that many users already have downloaded jQuery from Google or Microsoft when visiting another site, thus it will be loaded from cache when they visit your site, which leads to faster loading time.
- However, using CDN is not recommended for development, since the editor's intellisense doesn't work well with CDNs

jQuery Syntax

- With jQuery you select (query) HTML elements and perform some actions on them
- The basic syntax is: **$(“*selector*”).*action*()**
  - The $ sign stands for the jQuery function
  - A s*elector* is used to query or find HTML elements
    - All CSS selectors are supported, and jQuery adds some more selectors of its own
  - $(“*selector”*) creates a new object that contains the selected elements
  - The jQuery *action*() is then performed on the selected element(s)
- Examples:
  - $("p").hide() - hides all <p> elements
  - $(".test").hide() - hides all elements with class="test"
  - $("#test").hide() - hides the element with id="test“
  - $(this).hide() - hides the current element

jQuery First Example

```
<!DOCTYPE html>
<html>
<head>
    <title>jQuery Example</title>
</head>
<body>
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Third paragraph</p>

    <script src="jquery-3.3.1.js"></script>
    <script>
        $("p").html("Hello from jQuery").css("color", "blue");
    </script>
</body>
</html>
```



Select all paragraphs

Change the innerHTML of all selected elements

jQuery supports method chaining

Change CSS properties of selected elements

jQuery Selectors

- jQuery selectors allow you to select and manipulate HTML elements
- jQuery supports all CSS Selectors, and in addition, it has some own custom selectors
- **Element selector**
  - `$("p")` – all paragraph elements
  - `$("*")` – all elements in document
- **Class selector**
  - `$(".normal")` – all elements with **normal** class
- **ID selector**
  - `$("#nav")` – (first) element with the id **nav**
- **Group selector**
  - `$("p, li, .status, #nav")`

jQuery Selectors

- **Hierarchy selectors**
  - `$("#footer span")` – **Span** elements that are descendants of element with id **footer**
  - `$("div > span")` – **Span** elements that are immediate **children** of **div** elements
  - `$("h2 + span")` – **Span** elements that are immediately **preceded** by **h2** elements
  - `$("nav ~ span")` – **Span** elements that **following nav** elements of the **same level**
- **Attribute selectors**
  - `$("a[href]")` – **a** elements that **have** an **href** attribute
  - `$("a[href='test.html']")` – **a** elements whose **href** attribute value **equals** to **test.html**
  - `$("a[href!='do.asp']")` – **a** elements whose **href** attribute value **doesn't equal** to do.asp
  - `$("a[href^='admin']")` – **a** elements whose **href** attribute value **starts** with **admin**
  - `$("a[href*='public']")` – **a** elements whose **href** attribute value **contains public**
  - `$("a[href$='.pdf']")` – **a** elements whose **href** attribute value **ends** with **.pdf**

jQuery Selectors

- **Pseudo-selectors**
  - `$("div:eq(2)")` – Select the **3rd div** (0 based index)
  - `$("div:gt(1)")` – Select **div** elements following the **2nd**
  - `$("div:lt(3)")` – Select **div** elements up to the first **4th**
  - `$("div:first") / $("div:last")` – Select the **first / last div** element in the document
  - `$("div:even") / $("div:odd")` – Select the **even** (0,2,4) **/ odd** (1,3,5…) **div** elements by **index**
  - `$("div:contains('hello')")` – Select **div** elemets that contain the string "**hello**"
  - `$("div:has(h2)")` – Select **div** elements that contain **h2** element(s) as descendent
  - `$(":input:not(:text)")` – Select **input** elements that are not **textbox / textarea** elements
  - `$("div:visible") / $("div:hidden")` – Select **visible / hidden div** elements

Selecting Form Elements

- jQuery offers several pseudo-selectors that help find elements in forms
- These are especially helpful because it can be difficult to distinguish between form elements based on their state or type using standard CSS selectors
  - `$(":button, :checkbox, :file, :image, :password, :radio, :reset, :submit, :text")` – **Input** elements of the specified **type**
  - `$(":input")` – **Input** elements, including **select** & **textarea**
  - `$(":checked")` – **Checkbox & radio** elements that are **checked**
  - `$(":selected")` – Selected **option** elements
  - `$(":enabled") / $(":disabled")` – **Enabled / disabled** form elements

Testing the Selection

- Once you've made a selection, you'll often want to know whether you have anything to work with
- A common mistake is to use:

```
// Doesn't work!
if ($("div.foo")) {
    ...
}
```

- This won't work. When a selection is made using $(), an object is always returned, and objects always evaluate to true
- The best way to determine if there are any elements is to test the selection's **.length** property, which tells you how many elements were selected

```
// Testing whether a selection contains elements.
if ($("div.foo").length) {
    ...
}
```

Saving Selections

- If you've made a selection that you might need to make again, you should save the selection in a variable rather than making the selection repeatedly

```
let divs = $("div");
```

- Once the selection is stored in a variable, you can call jQuery methods on the variable just like you would have called them on the original selection
- Note that a selection only fetches the elements that are on the page at the time the selection is made
- If elements are added to the page later, you'll have to repeat the selection or otherwise add them to the selection stored in the variable

Refining and Filtering Selections

- Sometimes the selection contains more than what you're after
- jQuery offers several methods for refining and filtering selections:

```
$("div.foo").has("p");          // div.foo elements that contain <p> tags
$("h1").not(".bar");            // h1 elements that don't have a class of bar
$("ul li").filter(".current"); // unordered list items with class of current
$("ul li").first();             // just the first unordered list item
$("ul li").eq(5);               // the sixth list item
```

jQuery DOM Manipulation

- jQuery provides a set of methods to access and manipulate DOM elements and attributes:
  - **text()** – get or set the text content of selected elements
  - **html()** – get or set the HTML content of selected elements (including HTML markup)
  - **val()** – get or set the value of form elements
  - **attr()** – get or set the value of the specified HTML attribute
  - **width()** – get or set the width in pixels of the element
  - **height()** – get or set the height in pixels of the element

jQuery DOM Manipulation Example

- The following example demonstrates how to read the values of two input numbers and display their sum on a span:

```
<body>
    Num1: <input type="number" id="num1" value="10" /><br />
    Num2: <input type="number" id="num2" value="5" /><br />
    Sum: <span id="sum"></span>

    <script src="jquery-3.3.1.js"></script>
    <script>
        let num1 = Number($("#num1").val());
        let num2 = Number($("#num2").val());
        let sum = num1 + num2;

        $("#sum").html(sum);
    </script>
</body>
```

Exercise (1)

- Write jQuery code to change the hyperlink and the text of the following link to http://jquery.com and jQuery, respectively

```
<body>
    <a href="http://google.com">Google</a>
</body>
```

Exercise (2)

- Display the value of the selected option in the following cities list:

```
<select id="city">
    <option value="sydney">Sydney</option>
    <option value="melbourne">Melbourne</option>
    <option value="cromwell" selected>Cromwell</option>
    <option value="queenstown">Queenstown</option>
</select>
```

- Change the selected option to be “melbourne” instead of “cromwell”

CSS and Styling

- jQuery includes a handy way to get and set CSS properties of elements:

```
// Getting CSS properties
console.log($("h1").css("fontSize"));
console.log($("h1").css("font-size")); // Also works
```

```
// Setting CSS properties
$("h1").css("fontSize", "50px"); // Setting an individual property

// Setting multiple properties
$("h1").css({
    fontSize: "50px",
    color: "red"
});
```

Using CSS Classes for Styling

- The css() method should generally be avoided as a setter in production-ready code, because it's generally best to keep presentational information out of JavaScript code
- Instead, write CSS rules for classes that describe the various visual states, and then change the class on the element

```
// Working with classes
let h1 = $("h1");

h1.addClass("big");
h1.removeClass("big");
h1.toggleClass("big");

if (h1.hasClass("big")) {

}
```

- Classes can also be useful for storing state information about an element, such as indicating that an element is selected

Dimensions

- jQuery offers a variety of methods for obtaining and modifying dimension and position information about an element

```
// Basic dimensions methods

// Sets the width of all <h1> elements
$("h1").width("50px");

// Gets the width of the first <h1> element
$("h1").width();

// Sets the height of all <h1> elements
$("h1").height("50px");

// Gets the height of the first <h1> element
$("h1").height();

// Returns an object containing position information for
// the first <h1> relative to its "offset (positioned) parent"
$("h1").position();
```

Exercise (3)

- Find all h1 elements that are children of a div element, change their background color to blue, and change their text color to white
  - Using the css() method
  - Using CSS classes

```
<body>
    <h1>abc</h1>
    <div>
        <h1>header-1</h1>
        <h1>header-2</h1>
    </div>
    <h1>xyz</h1>
</body>
```

Iterating over jQuery Objects

- jQuery provides a jQuery collection iterator: **.each()**
  - It iterates over each matched element in the collection and performs a callback on that object
  - The index of the current element in the collection is passed as an argument to the callback
  - The callback is fired within the context of the current matched element so the **this** keyword points to the current element
- For example, given the following markup:

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

- $.each() may be used to display the contents of each list item:

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

```
Item 1
Item 2
Item 3
```

DOM Traversal

- Once you've made an initial selection with jQuery, you can traverse deeper into what was just selected
- jQuery provides methods for selecting the **children**, the **parents**, and the **siblings** of each element in the wrapped set
- Most of these methods accept an optional jQuery selector argument for **filtering**
-

```
$("div").children("p")// Select p elements that are direct descendants of div elements
$("div").find("p")    // Select p elements that are descendants of div elements
$("div").prev()       // prev(elem)Select set of elements immediately precede div elements
$("div").next()       // next(elem)Select set of elements immediately follow div elements
$("div").siblings("p")// Select p elements that are siblings of div elements
$("div").parent()     // Select elements that are parents of div elements
```

- Be cautious when traversing long distances in documents – complex traversal assumes that document's structure remains the same, which is difficult to guarantee

Creating New Elements

- jQuery offers a trivial and elegant way to create new elements using the same **$()** method used to make selections:

```
// Creating new elements from an HTML string
$("<p>This is a new paragraph</p>");
$("<li class='new'>new list item</li>");
```

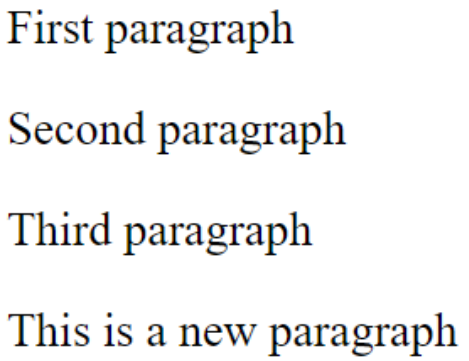
```
// Creating a new element with an attributes object
$("<a/>", {
    html: "This is a <strong>new</strong> link",
    "class": "new",
    href: "foo.html"
});
```

  - Note that in the attributes object above the property name “class” is quoted, since it is a reserved word

Creating New Elements

- There are several ways to add an element to the page once it's been created:
  - **append()** - Inserts the new element at the end of the selected element(s)
  - **prepend()** - Inserts the new element at the beginning of the selected element(s)
  - **after()** - Inserts the new element after the selected element(s)
  - **before()** - Inserts the new element before the selected elements(s)

```
<body>
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Third paragraph</p>

    <script src="jquery-3.3.1.js"></script>
    <script>
        let newParagraph = $("<p>This is a new paragraph</p>");
        $("body").append(newParagraph);
    </script>
</body>
```



Creating New Elements

- You can also create an element as you're adding it to the page, but note that in this case you don't get a reference to the newly created element:

```
<body>
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Third paragraph</p>

    <script src="jquery-3.3.1.js"></script>
    <script>
        $("body").append("<p>This is a new paragraph</p>");
    </script>
</body>
```

First paragraph

Second paragraph

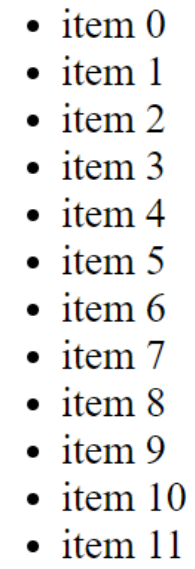
Third paragraph

This is a new paragraph

Creating New Elements

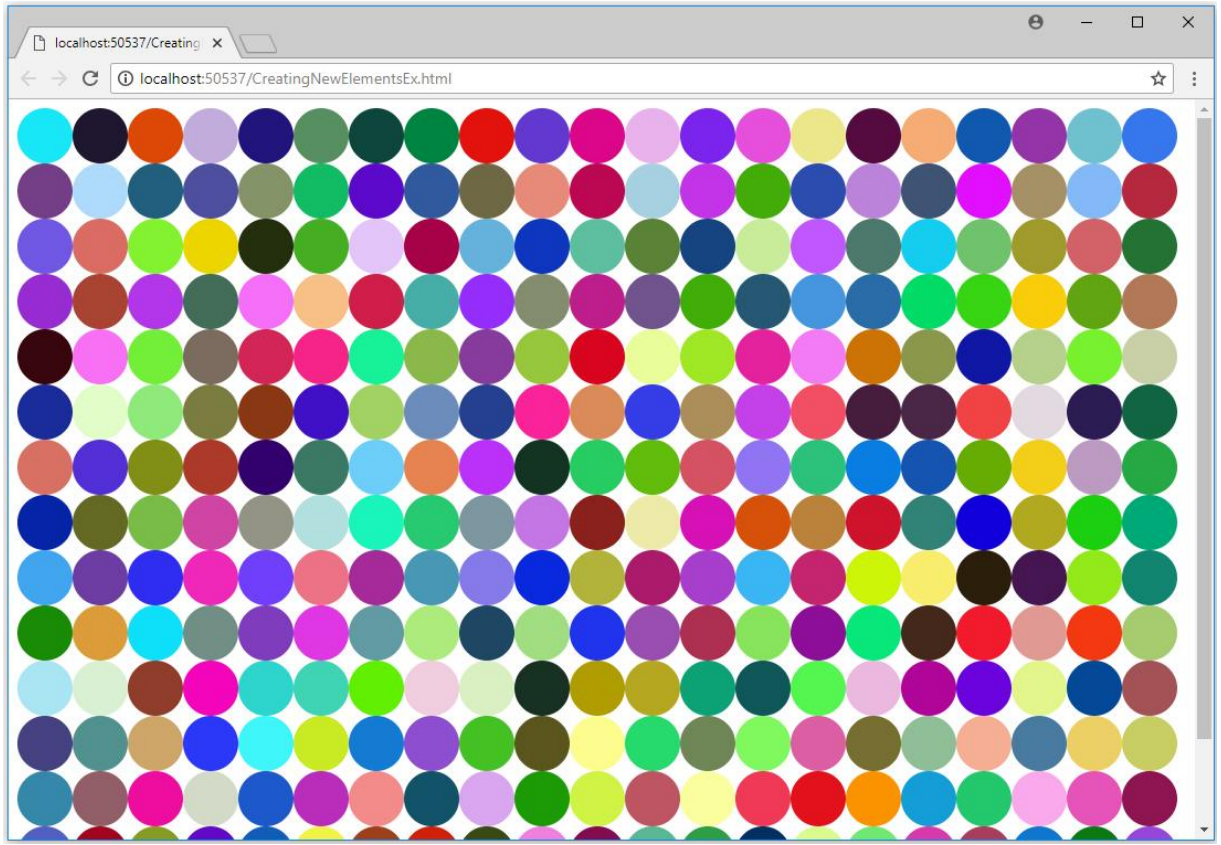
- The syntax for adding new elements to the page is easy, so it's tempting to forget that there's a huge performance cost for adding to the DOM repeatedly
- If you're adding many elements to the same container, you'll want to concatenate all the HTML into a single string, and then append that string to the container instead of appending the elements one at a time

```
<ul id="myList"></ul>

<script>
    let myItems = [];
    let myList = $("#myList");

    for (var i = 0; i < 100; i++) {
        myItems.push("<li>item " + i + "</li>");
    }

    myList.append(myItems.join(""));
</script>
```



Exercise (4)

- Use jQuery to generate 300 circles with random colors and draw them on the page
  - Hint: you can use with border-radius to represent each circle



Removing Elements

- There are two ways to remove elements from the page:
  - **.remove()** - permanently removes the selection from the page
  - **.detach()** - maintains the data and events associated with the selection, so you can restore the selection to the page at a later time
    - .detach() is extremely valuable if you are doing heavy manipulation on an element
    - In that case, it's beneficial to .detach() the element from the page, work on it in your code, then restore it to the page when you're done
    - This limits expensive "DOM touches" while maintaining the element's data and events
- If you want to leave the element on the page but remove its contents, you can use **.empty()** to dispose of the element's inner HTML

Moving Elements

- There are generally two approaches for moving elements around:
  - Place the selected element(s) relative to another element
  - Place an element relative to the selected element(s)
- For example, jQuery provides .insertAfter() and .after()
  - .insertAfter() method places the selected element(s) after the element given as an argument
  - .after() method places the element provided as an argument after the selected element
- Several other methods follow this pattern: .insertBefore() and .before(), .appendTo() and .append(), and .prependTo() and .prepend()

```
// Make the first list item the last list item:
var li = $("#myList li:first").appendTo("#myList");

// Another approach to the same problem:
$("#myList").append($("#myList li:first"));

// Note that there's no way to access the list item that we moved,
// as this returns the list itself.
```

jQuery Events

- jQuery offers convenience methods for most native browser events:

blur	change	click	dblclick	error	focus
focusin	focusout	keydown	keypress	keyup	load
mousedown	mouseenter	mouseleave	mousemove	mouseout	mouseover
mouseup	resize	scroll	select	submit	unload

- These methods are shorthand for jQuery's **.on()** method
  - The on() method is useful for binding the same handler function to multiple events, when you want to provide data to the event handler, or when you are working with custom events

```
// Event setup using a convenience method
$("p").click(function () {
    alert("You clicked a paragraph!");
});

// Equivalent event setup using the .on() method
$("p").on("click", function () {
    alert("You clicked a paragraph!");
});
```

Tearing Down Event Listeners

- To remove an event listener, you use the **.off()** method and pass in the event type to off
- If you attached a named function to the event, then you can isolate the event tear down to just that named function by passing it as the second argument

```
// Tearing down all click handlers on a selection
$("p").off("click");
```

```
// Tearing down a particular click handler, using a reference to the function
let foo = function () { console.log("foo"); };
let bar = function () { console.log("bar"); };

$("p").on("click", foo).on("click", bar);
$("p").off("click", bar); // foo is still bound to the click event
```

Setting Up Events to Run Only Once

- Sometimes you need a particular handler to run only once - after that, you may want no handler to run, or you may want a different handler to run
- jQuery provides the **.one()** method for this purpose

```
// Switching handlers using the .one() method
$("p").one("click", firstClick);

function firstClick() {
    console.log("You just clicked this for the first time!");

    // Now set up the new handler for subsequent clicks;
    // omit this step if no further click responses are needed
    $(this).click(function () { console.log("You have clicked this before!"); });
}
```

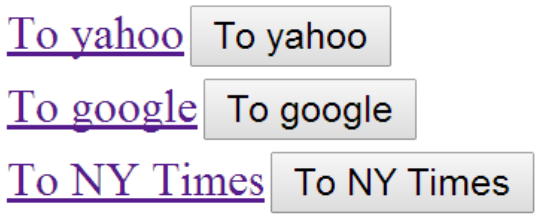
Event Helpers

- jQuery offers a few event-related helper functions that save you a few keystrokes
- For example, the **.hover()** method lets you pass one or two functions to be run when the mouseenter and mouseleave events occur on an element
  - If you pass one function, it will be run for both events
  - If you pass two functions, the first will run for mouseenter, and the second will run for mouseleave

```
// The hover helper function
$("#menu li").hover(function () {
    $(this).toggleClass("hover");
});
```

Exercise (5)

- You have a page with three links
- Using jQuery:
  - Add a button next to each link
  - The button's text should be equal to the text of the link
  - Clicking the button should change the window's location to the address of the link



Event Delegation

- Event delegation allows us to attach a single event listener, to a parent element, that will fire for all descendants matching a selector, whether those descendants exist now or are added in the future
- For example, assume we have the following HTML page:

```
<html>
<body>
    <ul id="list">
        <li><a href="http://domain1.com">Item #1</a></li>
        <li><a href="/local/path/1">Item #2</a></li>
        <li><a href="/local/path/2">Item #3</a></li>
        <li><a href="http://domain4.com">Item #4</a></li>
    </ul>
</body>
</html
```

- When an anchor in our #list group is clicked, we want to log its text to the console

Event Delegation

- Normally we could bind to the click event of each anchor using the .on() method:

```
// Attach a directly bound event handler
$("#list a").on("click", function (evt) {
    evt.preventDefault();
    console.log($(this).text());
});
```

- While this works perfectly fine, consider what happens when we add a new anchor after having already bound the above listener:

```
// Add a new element on to our existing list
$("#list").append("<li><a href='http://newdomain.com'>Item #5</a></li>");
```

- If we were to click our newly added item, nothing would happen
- This is because direct events are only attached to elements at the time the .on() method is called

Event Delegation

- To fix this problem, we can take advantage of event bubbling or propagation
- Any time one of our anchor tags is clicked, a *click* event is fired for that anchor, and then bubbles up the DOM tree, triggering each of its parent click event handlers:
  - <a>
  -
  - <ul #list>
  - <body>
  - <html>
  - *document* root
- This means that anytime you click one of our bound anchor tags, you are effectively clicking the entire document body!

Event Delegation

- Since we know how events bubble, we can create a **delegated event**:

```
// Attach a delegated event handler
$("#list").on("click", "a", function (evt) {
    evt.preventDefault();
    console.log($(this).text());
});
```

  - Notice how we have moved the **a** part from the selector to the second parameter position of the .on() method
  - When a click event occurs, the handler checks to see if the triggering element for that event matches the second parameter, and if it does - our anonymous function will execute
  - We've now attached a single click event listener to our <ul> that will listen for clicks on its descendant anchors, instead of attaching an unknown number of directly bound events to the existing anchor tags only

Exercise (6)

- You are given the following HTML table:

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada

- Write jQuery code to add a Delete button next to each row
- Clicking the Delete button should remove the corresponding row from the table
- You can use the HTML code on the next slide as your starting page

Exercise (6)

```
<html>
<head>
    <style>
        table {
            font-family: arial, sans-serif;
            border-collapse: collapse;
        }

        td, th {
            border: 1px solid #dddddd;
            text-align: left;
            padding: 5px;
        }

        tr:nth-child(even) {
            background-color: #dddddd;
        }
    </style>
</head>
<body>
    <table>
        <thead>
            <tr>
                <th>Company</th>
                <th>Contact</th>
                <th>Country</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Alfreds Futterkiste</td>
                <td>Maria Anders</td>
                <td>Germany</td>
            </tr>
            <tr>
                <td>Centro comercial Moctezuma</td>
                <td>Francisco Chang</td>
                <td>Mexico</td>
            </tr>
            <tr>
                <td>Ernst Handel</td>
                <td>Roland Mendel</td>
                <td>Austria</td>
            </tr>
            <tr>
                <td>Island Trading</td>
                <td>Helen Bennett</td>
                <td>UK</td>
            </tr>
            <tr>
                <td>Laughing Bacchus Winecellars</td>
                <td>Yoshi Tannamuri</td>
                <td>Canada</td>
            </tr>
        </tbody>
    </table>
</body>
```

Data Methods

- There's often data about an element you want to store with the element
- In plain JavaScript, you might do this by adding a property to the DOM element, but you'd have to deal with memory leaks in some browsers
- jQuery offers a straightforward way to store data related to an element, and it manages the memory issues for you

```
// Storing and retrieving data related to an element
$("#myDiv").data("keyName", { foo: "bar" });

let obj = $("#myDiv").data("keyName"); // Returns { foo: "bar" }
console.log(obj);
```

- Any kind of data can be stored on an element

jQuery Effects

- jQuery makes it trivial to add simple effects to your page
- Effects can use the built-in settings, or provide a customized duration
- You can also create custom animations of arbitrary CSS properties

Showing and Hiding Content

- jQuery can show or hide content instantaneously with **.show()** or **.hide()**:

```
// Instantaneously hide all paragraphs
$("p").hide();

// Instantaneously show all divs that have the hidden style class
$("div.hidden").show();
```

- When jQuery hides an element, it sets its CSS display property to none
  - This means the content will have zero width and height
- You can also tell .show() and .hide() to use animation in a couple of ways:
  - Pass in an argument of 'slow', 'normal', or 'fast':

```
// Slowly hide all paragraphs
$("p").hide("slow");
```

  - Pass the desired duration in milliseconds:

```
// Hide all paragraphs over two seconds
$("p").hide(2000);
```

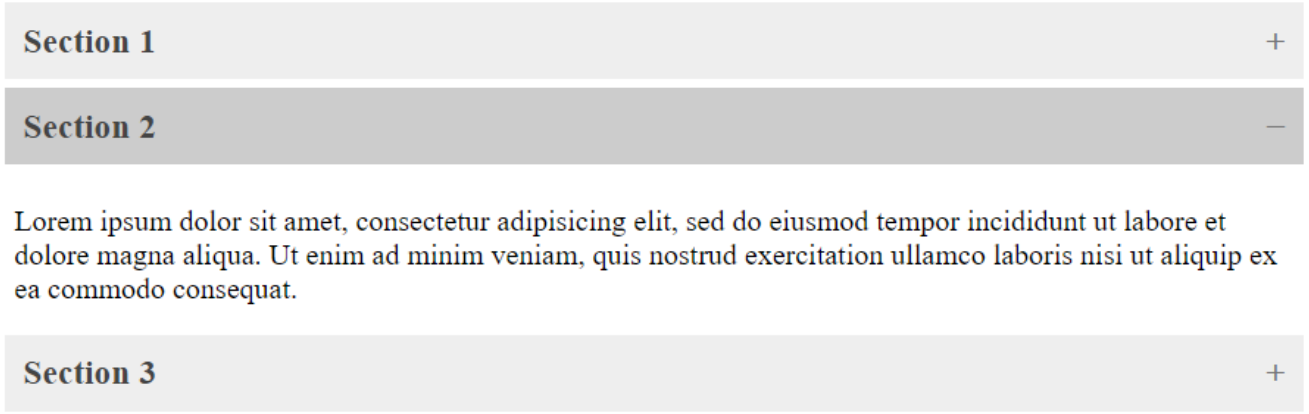
Fade and Slide Animations

- .show() and .hide() use a combination of slide and fade effects when showing and hiding content in an animated way.
- If you would rather show or hide content with one effect or the other, there are additional methods that can help:
- **.slideDown()** and **.slideUp()** show and hide content, using only a slide effect
  - These methods involve rapidly making changes to an element's CSS height property
- **.fadeIn()** and **.fadeOut()** show and hide content, by means of a fade animation
  - Fade animations involve rapidly making changes to an element's CSS opacity property

```
// Hide all paragraphs using a slide up animation over 0.8 seconds
$("p").slideUp(800);

// Show all hidden divs using a fade in animation over 0.75 seconds
$("div.hidden").fadeIn(750);
```

Exercise (7)

- In this exercise you will create an accordion (collapsible content)
- Accordions are useful when you want to toggle between hiding and showing large amount of content:



- Write jQuery code to create the accordion effect, based on the HTML given on the next slide

Exercise (7)

- Start with the following HTML

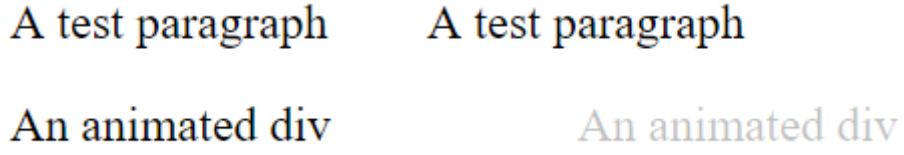
```
<head>
     <style>
        .accordion {
            background-color: #eee;
            color: #444;
            cursor: pointer;
            padding: 10px;
            margin: 0 0 5px 0;
        }

        .accordion:after {
            content: '\002B';
            color: #777;
            font-weight: bold;
            float: right;
        }

         .active, .accordion:hover {
            background-color: #ccc;
        }

        .active:after {
             content: "\2212";
        }
```

```
        .panel {
                padding: 0 5px;
                display: none;
                overflow: hidden;
            }
    </style>
</head>
<body>
    <h3 class="accordion">Section 1</h3>
    <div class="panel">
        <p>Lorem ipsum ...</p>
    </div>

    <h3 class="accordion">Section 2</h3>
    <div class="panel">
        <p>Lorem ipsum ...</p>
    </div>

    <h3 class="accordion">Section 3</h3>
    <div class="panel">
        <p>Lorem ipsum ...</p>
    </div>
</body>
```

Custom Effects with .animate()

- jQuery allows you to animate arbitrary CSS properties via the **.animate()** method
- You can animate to a set value, or to a value relative to the current value

```
$("div.anim").animate(
    {
        left: "+=50",
        opacity: 0.25
    },

    // Duration (in milliseconds)
    1000,

    // Callback to invoke when the animation is finished
    function () {
        console.log("done!");
    }
);
```



jQuery EcoSystem

- Since its inspection, jQuery has created an entire ecosystem consisting of companion libraries and other projects such as these:
  - **jQuery UI** – A library consisting of a set of user interface interactions, effects, widgets, and themes to help you create amazing user interfaces
  - **jQuery Mobile** – An HTML5-based user interface system for all popular mobile device platforms, to help you create beautiful designs for mobile Devices
  - **QUnit** – A JavaScript unit-testing framework used by all the other jQuery projects
  - **Plugins** – The myriad of other plugins spread across the web that people have created to cover those use cases not covered by jQuery or to improve its functionalities

jQuery Plugins

- A plug-in is piece of code written in a standard JavaScript file
- Plugins provide useful jQuery methods/components which can be used along with jQuery library methods
- You could consider each method that comes with the jQuery core a plugin
- There are plenty of jQuery plug-in available which you can download from https://jquery.com/plugins
- Writing your own custom plugins allows you to:
  - Encapsulate your code for reuse purposes
  - Public distribution
  - Maintain chainability
  - Prevent namespace clashing

Creating a Custom Plugin

- A plugin is defined by adding a method to the jQuery prototype (jQuery.fn)
- Encapsulate the plugin in a closure
  - To hide inner functions
- The plugin is passed the jQuery object that contains the DOM selection
  - It is accessible using the **this** keyword (not $(this))
- Return this to maintain chainability
- Save into a JS file `jquery.pluginname.js`
- Finally to use your plugin you would write:

```
(function ($) {
    $.fn.pluginname = function () {
        // Implement your plugin here
        return this;
    };
})(jQuery);
```

```
$("selector").pluginname();
```

jQuery Custom Plugin Example

```
(function ($) {
    $.fn.makeEqualSize = function () {
        var maxHeight = 0;
        var maxWidth = 0;

        this.each(function () {
            var currHeight = $(this).height();
            var currWidth = $(this).width();

            if (currHeight > maxHeight)
                maxHeight = currHeight;
            if (currWidth > maxWidth)
                maxWidth = currWidth;
        });

        this.each(function () {
            $(this).height(maxHeight);
            $(this).width(maxWidth);
        });
        return this;
    };
})(jQuery);
```

```
<html>
<head>
    <title>jQuery Plugin Example</title>
</head>
<body>
    <img src="images/minion.png"/>
    <img src="images/sam.png" />
    <img src="images/exit.png" />
    <script src="jquery-3.2.1.js"></script>
    <script src="scripts/jquery.makeEqualSize.js"></script>
    <script>
        $("img").makeEqualSize();
    </script>
</body>
</html>
```