

# Introduction to jQuery Training



with examples and  
hands-on exercises

---

**WEBUCATOR**

Copyright © 2024 by Webucator. All rights reserved.

No part of this manual may be reproduced or used in any manner without written permission of the copyright owner.

**Version:** 1.2.1

## The Authors

### ***Chris Minnick***

Chris Minnick, the co-founder of WatzThis?, has overseen the development of hundreds of web and mobile projects for customers from small businesses to some of the world's largest companies. A prolific writer, Chris has authored and co-authored books and articles on a wide range of Internet-related topics including HTML, CSS, mobile apps, e-commerce, e-business, Web design, XML, and application servers. His published books include Adventures in Coding, JavaScript For Kids For Dummies, Writing Computer Code, Coding with JavaScript For Dummies, Beginning HTML5 and CSS3 For Dummies, Webkit For Dummies, CIW E-Commerce Designer Certification Bible, and XHTML.

### ***Nat Dunn (Editor)***

Nat Dunn is the founder of Webucator ([www.webucator.com](http://www.webucator.com)), a company that has provided training for tens of thousands of students from thousands of organizations. Nat started the company in 2003 to combine his passion for technical training with his business expertise, and to help companies benefit from both. His previous experience was in sales, business and technical training, and management. Nat has an MBA from Harvard Business School and a BA in International Relations from Pomona College.

Follow Nat on Twitter at @natdunn and Webucator at @webucator.

## Class Files

Download the class files used in this manual at

<https://static.webucator.com/media/public/materials/classfiles/JQY111-1.2.1-introduction-to-jquery-training.zip>.

## Errata

Corrections to errors in the manual can be found at <https://www.webucator.com/books/errata/>.

# Table of Contents

LESSON 1. Getting Started with jQuery.....	1
jQuery in the 2020s.....	1
Our Approach.....	5
<b>Exercise 1: Getting Bootstrap.....</b>	8
<b>Exercise 2: Reviewing the Vanilla JavaScript Code.....</b>	9
<b>Exercise 3: Getting Started with jQuery.....</b>	10
LESSON 2. The jQuery Function and Selectors.....	13
The Document is Ready.....	14
<b>Exercise 4: Waiting for the Load Event.....</b>	16
jQuery Selectors.....	18
Filtering.....	23
Tree Traversal.....	24
Caching jQuery Objects.....	26
<b>Exercise 5: Playing with Selectors.....</b>	27
Chaining.....	28
Utility Functions.....	29
LESSON 3. jQuery Manipulation.....	33
Getter and Setter Methods.....	33
<b>Exercise 6: Getter and Setter Methods Practice.....</b>	35
Setting and Adding Content.....	35
<b>Exercise 7: Setting and Adding Content.....</b>	38
Copying and Removing Content.....	38
<b>Exercise 8: Setting and Adding Content.....</b>	40
event.target.....	40
Properties vs. Attributes.....	41
Shopping List Application.....	42
<b>Exercise 9: Logging.....</b>	44
<b>Exercise 10: Adding EventListeners.....</b>	47
<b>Exercise 11: Adding Items to the List.....</b>	51
<b>Exercise 12: Dynamically Adding Remove Buttons to the List Items.....</b>	53
<b>Exercise 13: Removing List Items.....</b>	55
<b>Exercise 14: Preventing Duplicates and Zero-length Product Names.....</b>	57

LESSON 4. jQuery Forms and Events.....	61
Listening for Events.....	61
Triggering Events.....	69
Delegating Events.....	70
<b>Exercise 15: Event Delegation.....</b>	<b>75</b>
LESSON 5. jQuery Effects.....	77
Display Effects.....	77
Fading Effects.....	78
<b>Exercise 16: Waiting for Fading to Finish.....</b>	<b>83</b>
Sliding Effects.....	84
Animating CSS Properties.....	85
LESSON 6. Ajax and jQuery.....	87
What is Ajax?.....	87
jQuery's Ajax Methods.....	90
<b>Exercise 17: Form Validation with Ajax.....</b>	<b>91</b>
LESSON 7. Converting from jQuery to JavaScript.....	95
Why Convert from jQuery to JavaScript?.....	95
<b>Exercise 18: Getting Ready.....</b>	<b>96</b>
<b>Exercise 19: Converting the Common Functions.....</b>	<b>107</b>
<b>Exercise 20: Converting the Config View Functions.....</b>	<b>110</b>
<b>Exercise 21: Convert the Game View Functions.....</b>	<b>113</b>
LESSON 8. Converting from JavaScript to jQuery.....	121
Why Convert to jQuery?.....	121
<b>Exercise 22: Getting Ready.....</b>	<b>122</b>
<b>Exercise 23: Converting the Common Functions.....</b>	<b>133</b>
<b>Exercise 24: Converting the Config View Functions.....</b>	<b>136</b>
<b>Exercise 25: Convert the Game View Functions.....</b>	<b>139</b>
LESSON 9. Review of Mathificent jQuery Code.....	147

# LESSON 1

## Getting Started with jQuery

**EVALUATION COPY: Not to be used in class.**

### Topics Covered

- A little history.
- jQuery in the 2020s.
- Our project.
- Your first jQuery code.

### Introduction

A lot has changed since jQuery was introduced in 2006. In this lesson, we provide a little history, and explain why it is still valuable to learn jQuery today. We'll also introduce you to the main project you will be working on, and whet your appetite with a tiny snippet of jQuery code.

**EVALUATION COPY: Not to be used in class.**



### 1.1. jQuery in the 2020s

When John Resig created jQuery in January, 2006, the world of browsers was a scary place. Microsoft Internet Explorer held 90% of the market, Firefox and Safari were just beginning to gain market share, and Google Chrome, which today is far and away the most popular browser, wouldn't appear for almost three years.<sup>1</sup> Different browsers followed standards to different degrees, and the standards themselves were still evolving rapidly. The W3C released the first draft specification of the XMLHttpRequest object, the original backbone of Ajax, in April, 2006, but all major browsers had already supported some degree of Ajax functionality for several years. They just did it in different ways.

---

1. See [https://en.wikipedia.org/wiki/Usage\\_share\\_of\\_web\\_browsers](https://en.wikipedia.org/wiki/Usage_share_of_web_browsers) for browser usage statistics.

This situation resulted in a nightmare for web developers, many of whom despised JavaScript. Code had to be branched with conditional statements that checked the browser for feature support:

```
if (thisFeatureIsSupported()) {  
    doItThisWay();  
} else if (thisOtherFeatureIsSupported()) {  
    doItThisOtherWay();  
} else {  
    alert('Sorry, this feature is not supported by your browser.');//  
}
```

It was ugly!

While many JavaScript libraries attempted to solve this problem, jQuery quickly emerged as the leader, and it remains the most widely used JavaScript library today, in large part because it is baked in to so many existing websites – almost 75% of the top 10 million!<sup>2</sup>

### ❖ 1.1.1. Problems jQuery Solves

jQuery was created to solve three major problems:

1. Allow developers to write code in one way that worked on all major browsers.
2. Make it easier to write Ajax applications.
3. Make it easier to locate, modify, delete, and add elements to a web page via JavaScript.

On top of this, jQuery provided neat, easy-to-implement effects to make it easier for developers to make snazzier web pages.

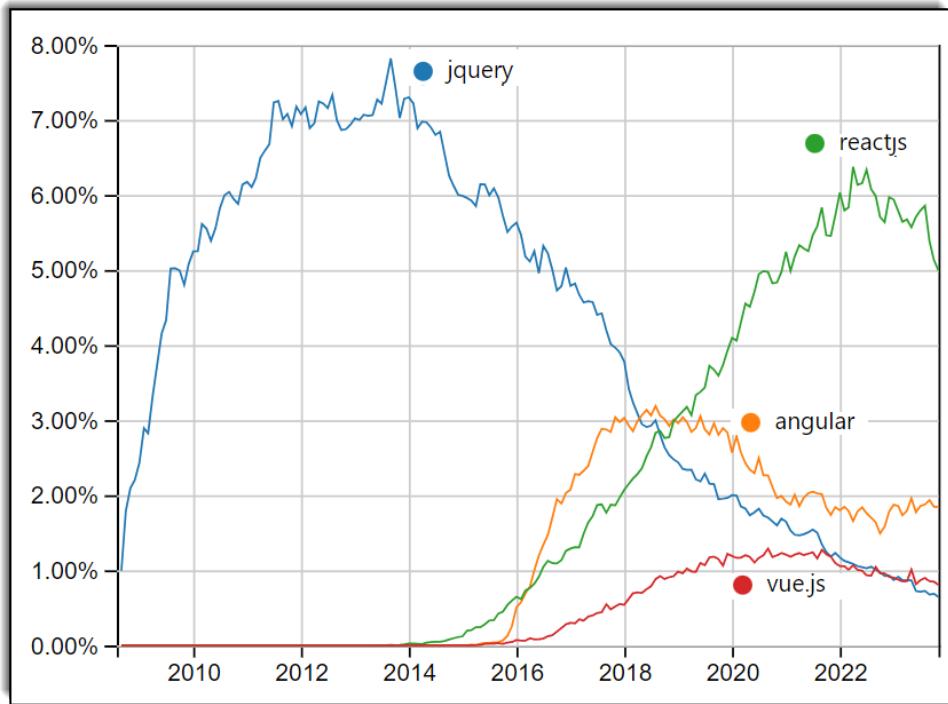
### ❖ 1.1.2. The Situation Today

Today, most of the original problems that jQuery addressed have been solved by improvements to the JavaScript language and the agreement by web browser makers on a standardized version of JavaScript. As the following charts show, jQuery has largely fallen out of favor:

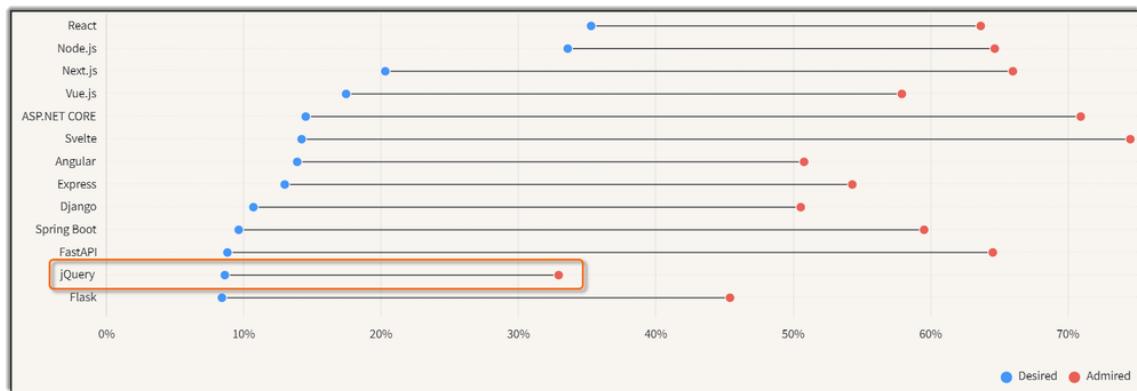
- The percentage of questions asked on Stack Overflow about jQuery has fallen dramatically over the last several years (<https://insights.stackoverflow.com/trends?tags=jquery%2Cangular%2Creactjs%2Cvue.js>):

---

2. <https://w3techs.com/technologies/details/js-jquery>



- jQuery is no longer among the most admired JavaScript frameworks.<sup>3</sup>



### ❖ 1.1.3. The Death of jQuery UI and jQuery Mobile

Two popular offshoots of jQuery were jQuery UI and jQuery Mobile, but they are both all but dead. The following screenshot shows a blog post from December 21, 2017:

3. <https://survey.stackoverflow.co/2023/#technology-most-popular-technologies>

## The Future of jQuery UI and jQuery Mobile

Posted on December 21, 2017 by Alex Schmitz

The last few years have been difficult for the jQuery UI and jQuery Mobile projects. The projects have suffered from lack of resources and funding and loss of contributors due to a variety of factors. These combined factors have nearly stopped development on both projects. To remedy this situation we have decided to make some changes in the projects' teams in addition to how they work.

The final ominous sentence of the post is “jQuery UI and jQuery Mobile rely on contributions from the community and can only continue to exist with your help!”

No help has come.

### ❖ 1.1.4. Dumped by Bootstrap

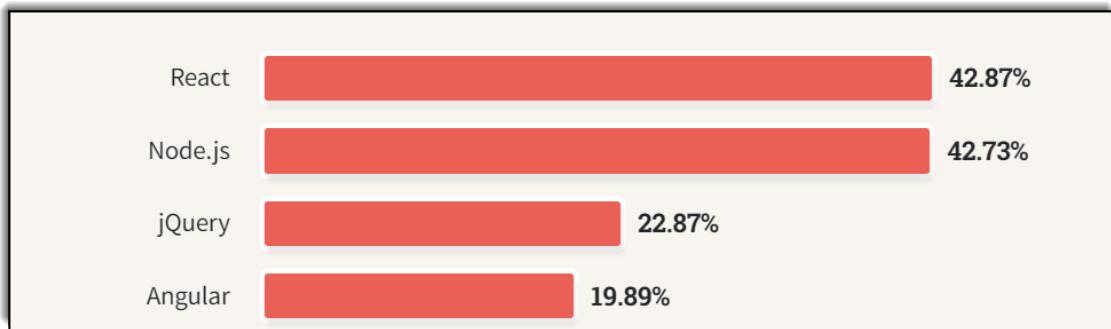
Another bad sign for jQuery is that it has been dropped from Bootstrap 5, the hugely popular HTML, CSS, and JavaScript library.

### ❖ 1.1.5. So, Why Learn jQuery?

Given that jQuery appears to be on its way out, why should you learn it today?

There are several reasons that make it still worth learning jQuery today:

1. jQuery is still among the most-used web framework by professional developers:<sup>4</sup>



2. As stated earlier, three out of every four large websites uses jQuery. This code will have to be maintained and/or converted to newer frameworks for years to come.
3. If you know modern JavaScript, jQuery is easy to learn, because much of the current JavaScript standard has been influenced by jQuery.

4. <https://survey.stackoverflow.co/2023/#most-popular-technologies-language-prof>

4. jQuery is still amazingly effective at making otherwise complex tasks simple. Its bells and whistles are still impressive.

Though professional developers purport to dread jQuery, we doubt that it is jQuery itself that they dread. We expect that number is high because most jQuery development these days is likely to be maintenance work or rewrites of old websites. This kind of work is simply less fun than building new sites. Be that as it may, there still seems to be plenty of demand for jQuery skills. At the time of this writing, a search for “jQuery” on the job site indeed.com returns over 4,000 jobs in the United States,<sup>5</sup> while a search for “Vue”, one of the most-loved modern frameworks, returns only 2,400.<sup>6</sup>

**EVALUATION COPY: Not to be used in class.**

## 1.2. Our Approach

\*  
*Evaluation  
Copy*

We assume you know JavaScript quite well already. In particular, you should be familiar with traversing the DOM and handling events. You should also be familiar with CSS selectors. Our goal is to give you a solid understanding of the differences between standard JavaScript (or, what we refer to as *vanilla JavaScript*) and jQuery. You will see how jQuery simplifies JavaScript development and even enables some functionality that would be difficult to write without the help of jQuery.

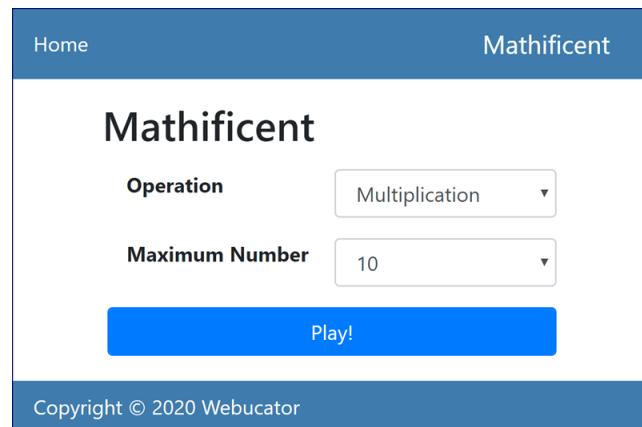
As part of your learning, you will have the option of converting a game called Mathificent<sup>7</sup> from vanilla JavaScript to jQuery or from jQuery to vanilla JavaScript, whichever you prefer. Or, if you want extra practice, you can do both – convert in one direction and then back again.

Mathificent contains the following three views:

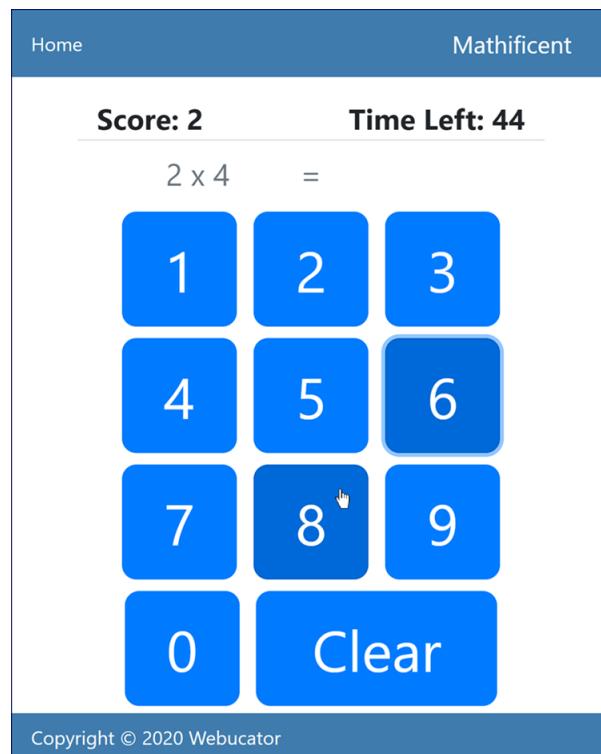
### 1. Config View

---

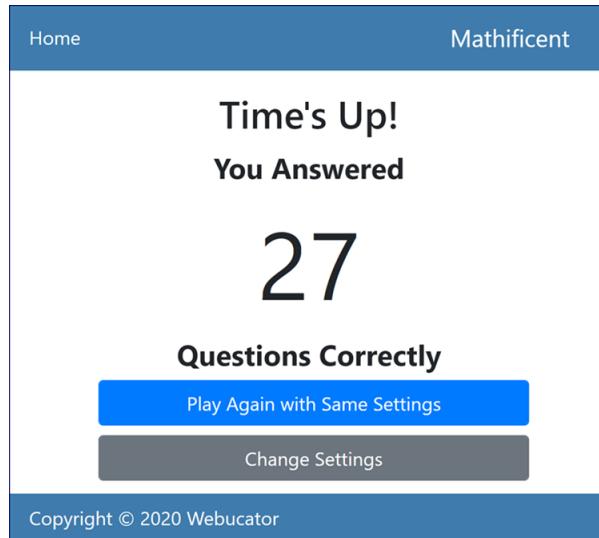
5. <https://www.indeed.com/jobs?q=jQuery&l=United+States>  
6. <https://www.indeed.com/jobs?q=Vue&l=United+States>  
7. <https://www.mathificent.com>



## 2. Game View



## 3. Times-up View



In the next couple of exercises, you will familiarize yourself with the Mathificent app.



# Exercise 1: Getting Bootstrap



20 to 30 minutes

The Mathificent app uses Bootstrap for styling.

1. Open `Exercises/javascript-to-jquery/index.html` in your editor.
2. Open `index.html` in the browser. It should open in the **Config** view, which should look like the screenshot shown earlier (see page 5).
3. Play a game of Mathificent to see how it works. Change the operation and the maximum number and play another game.

## Content Delivery Networks

A content delivery network (CDN) is a network of servers that host commonly used files. While you can download Bootstrap and deliver it from your own server, it generally makes more sense to use the Bootstrap CDN. There is a good chance that your site visitors have visited a different site that uses Bootstrap too, in which case, the user's browser will use the local copy, making your site available more quickly.

## Exercise 2: Reviewing the Vanilla JavaScript Code

 30 to 60 minutes

---

Spend 30 minutes to an hour reviewing the three files that make up the Mathificent app:

-  Exercises/javascript-to-jquery/index.html
-  Exercises/javascript-to-jquery/styles/mathificent.css
-  Exercises/javascript-to-jquery/scripts/mathificent.js

Pay particular attention to the JavaScript file, which is well commented. Make sure that you have a clear understanding of how all the JavaScript is working. We will come back to Mathificent in a later lesson. In the meantime, your goal is to learn jQuery well enough to be able to convert this application from vanilla JavaScript to jQuery.

# Exercise 3: Getting Started with jQuery

⌚ 10 to 15 minutes

1. Open `Exercises/getting-started/hello-world.html` in your editor. Notice that JavaScript is used to replace the content of the `h1` element:

```
document.getElementsByTagName('h1')[0].innerHTML = 'Hello, Vanilla JS!';
```

2. Open the `hello-world.html` file in your browser and notice that it does indeed replace the `h1` content with “Hello, Vanilla JS!”
3. In your web browser, go to <https://code.jquery.com/>.
4. Click the **minified** link next to the latest version of jQuery:



5. Click the copy icon to copy the `<script>` tag:



6. Paste the copied `<script>` tag into the head of `hello-world.html`.
7. Replace the JavaScript for replacing the content of the `h1` element with:

```
jQuery('h1').text('Hello, jQuery!');
```

- A. `jQuery('h1')` gets all the `h1` elements. In this case, there is only one.
- B. The `text()` method sets the `innerText` of the element.

8. Open the `hello-world.html` file in your browser again. Notice that it now replaces the `h1` content with “Hello, jQuery!”

Congratulations! You have written your first jQuery code!



## Solution: Solutions/getting-started/hello-world.html

---

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.  <meta charset="UTF-8">
5.  <meta name="viewport" content="width=device-width, initial-scale=1">
6.  <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous"
7.         integrity="sha256-/xUj+30JU5yExlq6GSYGSk7tPXikynS7ogEvDej/m4=></script>
8.  <title>Getting Started</title>
9.  </head>
10. <body>
11.   <h1>Hello, world!</h1>
12.   <script>
13.     jQuery('h1').text('Hello, jQuery!');
14.   </script>
15. </body>
16. </html>
```



---

## Conclusion

In this lesson, you learned the history of jQuery and why it is still worth learning even though it is becoming less popular. You also learned how to get jQuery on a page and you wrote your first snippet of jQuery code.

# LESSON 2

## The jQuery Function and Selectors

**EVALUATION COPY: Not to be used in class.**

### Topics Covered

- The document is ready.
- Selectors.
- Caching jQuery Objects.
- Utility Functions.

### Introduction

At the heart of jQuery is jQuery Core. jQuery Core provides the base set of features and a way of working with them that forms the foundation for all of the functionality that has been built on jQuery over the many years.

As we go through the different parts of jQuery, always keep in mind that jQuery is a JavaScript library. This means that everything that can be done with jQuery can also be done with vanilla JavaScript. However, jQuery, which calls itself the “write less, do more” JavaScript library, provides a simple interface for accomplishing many of the most common tasks that JavaScript developers do.

In this lesson, you’ll learn about the most important functionality of jQuery as well as the foundational ideas and techniques of jQuery development.

**EVALUATION COPY: Not to be used in class.**

\*

## 2.1. The Document is Ready

In the *hello-world* demo in the previous lesson, we placed the `script` element at the end of the document, immediately before the closing `</body>` tag. Because browsers process code as they find it in the document, reading from the top down, we have to wait until the element loads before we can replace its content.

The same thing can be accomplished using the `load` event of the `window` object. In vanilla JavaScript, we do that with the `addEventListener()` method:

```
window.addEventListener('load', function() {
  document.getElementsByTagName('h1')[0].innerHTML = 'Hello, Vanilla JS!';
});
```

In this case, the contained code will not run until the full `window` object has loaded, so we can safely place the code wherever we want.

### ❖ 2.1.1. The `jQuery()` Method

jQuery accomplishes the same thing using the following code:

```
jQuery(function() {
  jQuery('h1').text('Hello, jQuery!');
})
```

Notice that the `jQuery()` method is used twice in the preceding code.

1. The first time, it is used to replace:

```
window.addEventListener('load', function() { ... });
```

The jQuery equivalent is:

```
jQuery(function() { ... });
```

2. The second time, it is used to replace:

```
document.getElementsByTagName('h1')
```

The jQuery equivalent is:

```
jQuery('h1')
```

Because the `jQuery()` method is used so commonly in jQuery, it has this simple alias: `$()`. Using the `$()` alias, the preceding code can be rewritten like this:

```
$(function() {  
    $('h1').text('Hello, jQuery!');  
})
```

```
document.ready()
```

If you work with jQuery code that someone else has written, you are very likely to come across the following:

```
$(document).ready(function() {  
    ...  
});
```

For years, this was the most common way of waiting for the document to load, but the `ready()` method is officially deprecated as of jQuery 3. While it still works, for new code you should use the following instead:

```
$(function() {  
    ...  
});
```

# Exercise 4: Waiting for the Load Event

 5 to 10 minutes

---

In this exercise, you will modify the `hello-world.html` file from earlier so that the change to the DOM doesn't change until the document loads.

1. Open `Exercises/jquery-function-and-selectors/hello-world.html` in your editor.
2. Move the `script` element from the bottom of the document into the `head` right below the other `script` element.
3. Modify the code, so that the following line of code doesn't run until the page loads:

```
jQuery('h1').text('Hello, jQuery!');
```

Be sure to use jQuery.



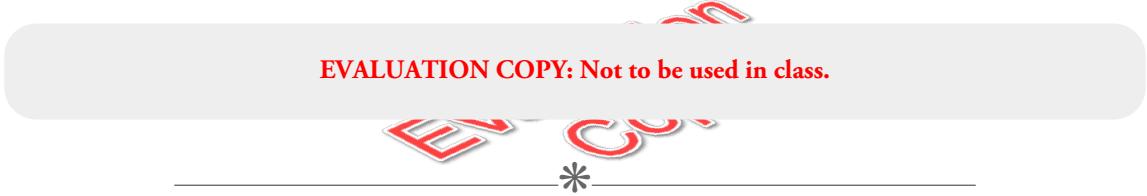
## Solution: Solutions/jquery-function-and-selectors/hello-world.html

---

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.  <meta charset="UTF-8">
5.  <meta name="viewport" content="width=device-width, initial-scale=1">
6.  <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous"
7.    integrity="sha256-/xUj+3OJU5yExlq6GSYGSKh7tPXikynS7ogEvDej/m4=></script>
8.  <script>
9.    $(function() {
10.      jQuery('h1').text('Hello, jQuery!');
11.    });
12.  </script>
13.  <title>Getting Started</title>
14.  </head>
15.  <body>
16.    <h1>Hello, world!</h1>
17.  </body>
18. </html>
```

---

**EVALUATION COPY: Not to be used in class.**



## 2.2. jQuery Selectors

There are several different ways to access elements on the page using vanilla JavaScript:

- `getElementById(id)` - returns a single `Element` Node with the passed-in `id` or `null` if no such element exists.
- `getElementsByClassName(className)` - returns an `HTMLCollection` of `Element` Nodes with the passed-in `className`.
- `getElementsByTagName(tagName)` - returns an `HTMLCollection` of `Element` Nodes with the passed-in `tagName`.
- `querySelectorAll(selector)` - returns a `NodeList` of `Element` Nodes matching the passed-in selector.
- `querySelector(selector)` - returns the first `Element` Node matching the passed-in selector.

jQuery uses `$(selector)`, which is essentially the same as `querySelectorAll(selector)`, for all of these. Some examples:

- `$('#main-nav')` – gets the element with the id “main-nav”.
- `('.col')` – gets all the elements with of the “col” class.
- `('h1')` – gets all the h1 elements.
- `('div#board>div.col')` – gets all the div elements of the “col” class within the div element with the id “board”.

Consider the following HTML document:

## Demo 2.1: Demos/jquery-function-and-selectors/lunch-order.html

---

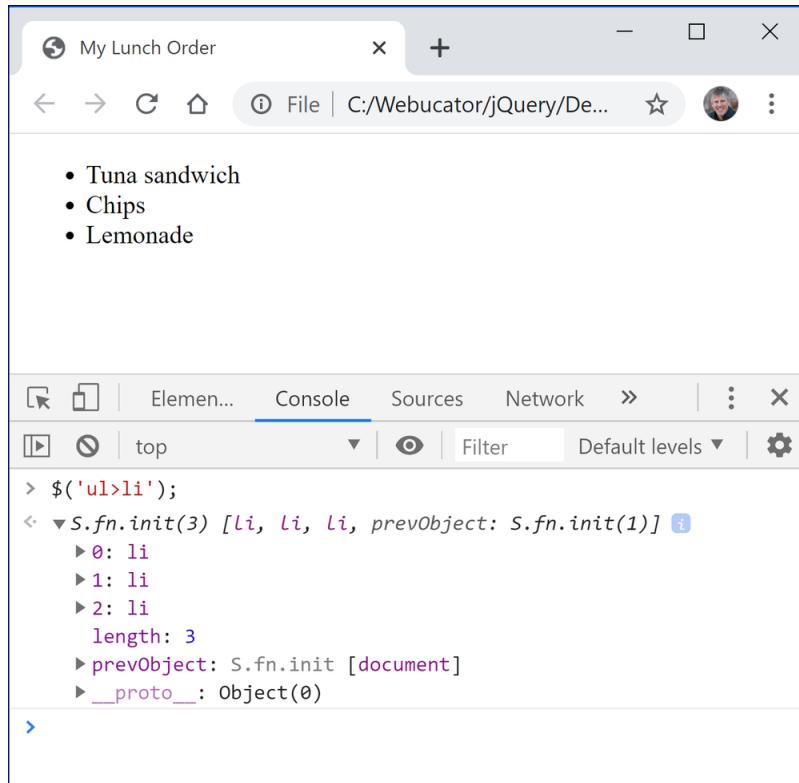
```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.  <meta charset="UTF-8">
5.  <meta name="viewport" content="width=device-width, initial-scale=1">
6.  <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous"
7.      integrity="sha256-/xUj+3OJU5yExlq6GSYCSHk7tPXikynS7ogEvDej/m4="></script>
8.  <title>My Lunch Order</title>
9.  </head>
10. <body>
11. <ul>
12.   <li>Tuna sandwich</li>
13.   <li>Chips</li>
14.   <li>Lemonade</li>
15. </ul>
16. </body>
17. </html>
```

---

To modify the list items in this document using jQuery, the first step is to select them. There are usually several ways to select nodes in a DOM tree. Here's one way to select the list items:

```
$(‘ul>li’)
```

This jQuery selector will create a jQuery object containing the three `li` elements that are direct children of the `ul` element. To see this in action, open the `lunch-order.html` demo in your browser and then open the Chrome DevTools Console. Type the preceding jQuery selector into the console and you'll see the resulting jQuery object that's returned:



Expand the jQuery object in the console to see each of the `li` elements, along with a lot of other data about your document.

An important consideration when using jQuery selectors is to be as specific as you can be so that you don't select elements that you didn't intend to select. This is especially important when you're writing jQuery to manipulate HTML documents that are dynamically generated or that may change in the future.

For example, what if our lunch order document were changed so that it contained a lunch order for every day of the week?

## Demo 2.2: Demos/jquery-function-and-selectors/lunch-order-by-day.html

---

-----Lines 1 through 10 Omitted-----

```
11. <div id="monday">
12.   <h2>Monday</h2>
13.   <ul>
14.     <li>Tuna sandwich</li>
15.     <li>Chips</li>
16.     <li>Lemonade</li>
17.   </ul>
18. </div>
19. <div id="tuesday">
20.   <h2>Tuesday</h2>
21.   <ul>
22.     <li>Tacos</li>
23.     <li>Chips</li>
24.     <li>Soda</li>
25.   </ul>
26. </div>
27. <div id="wednesday">
28.   <h2>Wednesday</h2>
29.   <ul>
30.     <li>Spaghetti</li>
31.     <li>Bread</li>
32.     <li>Iced Tea</li>
33.   </ul>
34. </div>
35. <div id="thursday">
36.   <h2>Thursday</h2>
37.   <ul>
38.     <li>Burger</li>
39.     <li>Fries</li>
40.     <li>Water</li>
41.   </ul>
42. </div>
43. <div id="friday">
44.   <h2>Friday</h2>
45.   <ul>
46.     <li>Pizza</li>
47.     <li>Beer</li>
48.   </ul>
49. </div>
```

-----Lines 50 through 51 Omitted-----

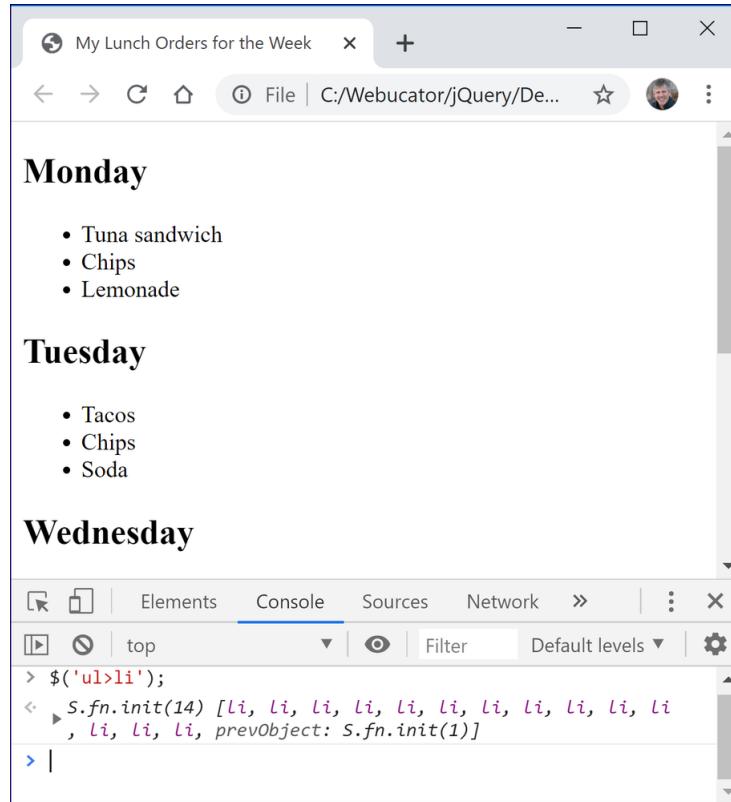
---

Evaluation  
Copy

## Code Explanation

---

If you use the same jQuery selector we used earlier (`$('.ul>li')`), you will select all the `li` items in the document, which may not be what you intended to do:



You can easily select the items for individual days by prefacing the `ul>li` selector with the `id` of the containing `div`:

```
$('#tuesday>ul>li')
```

The preceding selector will look for only the `li` elements that are direct children of `ul` elements that are inside of the element with an `id` value of “`tuesday`”. Try it out!

---

**EVALUATION COPY: Not to be used in class.**



## 2.3. Filtering

JQuery provides several “filter” methods whose job it is to modify a collection returned by the jQuery function. Some of the most useful filters are:

1. `eq()` – Takes a number as its argument and returns the element at the specified index.
2. `even()` – Returns the even-numbered elements in the collection.
3. `filter()` – Takes a selector or a function as its argument and returns a new collection of matching elements.
4. `first()` – Returns the first element in a collection.
5. `has()` – Reduces a collection to the elements that have a descendent matching the selector provided.
6. `is()` – Returns `true` if at least one of the elements in the collection matches the selector, element, or object passed to it.
7. `last()` – Returns the last element in the collection.
8. `odd()` – Returns the odd-numbered elements in the collection.
9. `slice()` – Accepts a range of element indices and returns a new collection with just those elements.

Open `Demos/jquery-function-and-selectors/filters.html` in your browser. Then, open the Chrome DevTools Console and experiment with jQuery filters by following the instructions on the page:

The screenshot shows a browser window with the title 'jQuery Filter Methods'. The page content includes instructions to open the JavaScript console and follow these steps to learn about jQuery's filter methods. The steps are numbered 1 through 9, each with a corresponding jQuery selector and its purpose:

1. Find the **first** li element:  
`\$('li').first().css('color', 'red');`
2. Find the **second** li element:  
`\$('li').eq(1).css('color', 'red');`
3. Find the **last** li element:  
`\$('li').last().css('color', 'red');`
4. Find all the even li elements:  
`\$('li').even().css('color', 'red');`  
Notice that it is the odd list items that become bold. That is because JavaScript is 0-based. So, the first li element is the zeroeth, which is even.
5. Find all the odd li elements:  
`\$('li').odd().css('color', 'red');`
6. Use the **filter()** function to find every third li element:  
`\$('li').filter(function(index) { return (index + 1) % 3 === 0; }).css('color', 'red');`
7. Use the **has()** method to find all li elements that have a **strong** descendent element:  
`\$('li').has('strong').css('color', 'red');`
8. Use the **is()** method to determine whether any of the elements in this page are h2 elements:  
`if(\$('body \*').is('h2')) { console.log('There is an h2 element.);} else { console.log('There are no h2 elements.);} `
9. Use **slice()** to find all but the first and last li elements:  
`\$('li').slice(1, \$('li').length-1).css('color', 'red');`

See <https://api.jquery.com/category/traversing/filtering/> for documentation on jQuery filters.

EVALUATION COPY: Not to be used in class.



## 2.4. Tree Traversal

Sometimes, it's useful to be able to find elements in HTML according to their position relative to another element. For these cases, jQuery provides a number of DOM traversal methods. Some of the most useful traversal methods are:

1. **children()** – Gets the children of the selected elements.
2. **find()** – Finds the descendants of the selected element and filters them using a selector.
3. **next()** – Gets the sibling element that immediately follows each of the selected elements.
4. **nextAll()** – Gets all of the following sibling elements of the selected element.
5. **parent()** – Gets the parent of each of the selected elements.

6. `parents()` – Gets the ancestors (all of the elements above the selected element in the DOM tree) of each of the selected elements.
7. `prev()` – Gets the sibling element that immediately precedes each of the selected elements.
8. `prevAll()` – Gets all of the preceding sibling elements of each of the selected elements.
9. `siblings()` – Gets all the siblings of the each of the selected elements.

The traversal methods can take a selector as an argument to limit the elements that are returned. For example, the following code would get all the `p` elements that follow an `h1` element:

```
$('h1').nextAll('p')
```

Open `Demos/jquery-function-and-selectors/traversing.html` in your browser. Then, open the Chrome DevTools Console and experiment with jQuery filters by following the instructions on the page:

The screenshot shows a browser window titled "jQuery Tree Traversal Methods". The address bar indicates the file is located at "File | nos/jquery-function-and-selectors/traversing.html". The main content area contains the following text:

## jQuery Tree Traversal Methods

Open the JavaScript console and follow these instructions to learn about jQuery's traversal methods.

After running each snippet of code, refresh the page.

1. Find all of the children of the `body` element:  
`$(‘body’).children().css(‘color’, ‘red’);`
2. Use the `find()` method to find the children elements of the `h1` element that are `p` elements:  
`$(‘h1’).find(‘p’).css(‘color’, ‘red’);`
3. Use the `next()` method to find the sibling element that immediately follows each of the `h1` elements:  
`$(‘h1’).next().css(‘color’, ‘red’);`
4. Use the `nextAll()` method to find all of the following siblings of the `h1` element:  
`$(‘h1’).nextAll().css(‘color’, ‘red’);`
5. Find the parent of the `ol` element:  
`$(‘ol’).parent().css(‘color’, ‘red’);`
6. Find all of the ancestors of the `ol` element:  
`$(‘ol’).parents().css(‘color’, ‘red’);`
7. Use the `prev()` method to find the siblings of the `h1` element that immediately precede it:  
`$(‘h1’).prev().css(‘color’, ‘red’);`
8. Use the `prevAll()` method to find all of the preceding siblings of the `ol` element:  
`$(‘ol’).prevAll().css(‘color’, ‘red’);`
9. Find all of the siblings of the `body` element:  
`$(‘body’).siblings().css(‘color’, ‘red’);`
10. Find all but the `p` that follow an `h1` element:  
`$(‘h1’).nextAll(‘p’).css(‘color’, ‘red’);`

Nice work!

See <https://api.jquery.com/category/traversing/tree-traversal/> for documentation on jQuery tree traversal.

**EVALUATION COPY: Not to be used in class.**



## 2.5. Caching jQuery Objects

Each time you call `$(selector)` it traverses the DOM looking for matches and returns a new jQuery object. When you need to make multiple modifications to the same element, it will speed things up if you cache the returned object by storing it in a constant. Consider the following code, which changes the text of all the `h1` elements and then changes the background color of the same elements:

```
$('h1').text('hello!');  
$('h1').css('background-color', 'blue');
```

The problem with the preceding code is that it's wasteful to locate the same DOM nodes for both of these changes. The solution is to assign the jQuery object to a constant and then use that constant for every subsequent operation that needs to be run on that set of nodes.

```
const allH1s = $('h1');  
allH1s.text('hello!');  
allH1s.css('background-color', 'blue');
```

# Exercise 5: Playing with Selectors

 20 to 30 minutes

1. Open `Exercises/jquery-function-and-selectors/mad-lib.html` in your editor. This file contains the text for a word replacement game, but the only way to replace the blanks with your words currently is by editing the HTML.
2. Inside the provided `<script>` tag, write a function called `madLib` that selects each `span` and replaces its inner text with appropriate words of your choice. In writing the function, try to stick with the following guidelines:
  - Don't make any changes to the HTML.
  - Write one jQuery statement for each replacement.
  - Make sure that each replaced word is unique.
  - Write the code so that you can rearrange the replacement statements without affecting the final text of the page.
  - Optimize your code by caching jQuery objects when possible.

Keep the file open in the browser as you're working through the exercise, so that you can refresh and see the effect of your changes to the code. The screenshot below shows the result of a partial solution:



## Solution: Solutions/jquery-function-and-selectors/mad-lib.html

---

```
-----Lines 1 through 9 Omitted-----  
10. <script>  
11.   function madLib() {  
12.     $('#title .singular-noun').text('Sandwich');  
13.     $('#first-para .verb-ing').text('brushing');  
14.     $('#first-para .singular-noun').text('car');  
15.     $('#first-para .adj').first().text('fluffy');  
16.     $('#first-para .adj').eq(1).text('dumb');  
17.     $('#second-para .plural-noun').first().text('bats');  
18.     $('#second-para .plural-noun').eq(1).text('goats');  
19.     $('#second-para .adj').text('goofy');  
20.     $('#second-para .singular-noun').first().text('plant');  
21.     $('#second-para .singular-noun').eq(1).text('boat');  
22.   }  
23.   $(madLib);  
24. </script>  
-----Lines 25 through 42 Omitted-----
```

EVALUATION COPY: Not to be used in class.

## 2.6. Chaining

Another way to optimize your jQuery code and reduce the amount of code you need to write is to use *chaining*. Chaining allows you to run multiple methods on a jQuery object with a single statement.

To use chaining, start with a jQuery function, such as this one:

```
$( 'h1' )
```

The returned value of this function is an object, which can take its own methods. Add a period to the end of the object and write your first method:

```
$( 'h1' ).text('hello!')
```

The preceding method will replace the inner text of the elements selected in the jQuery function with “hello!” Next, add another period to the end of this first method, followed by another method:

```
$(‘h1’).text(‘hello!’).css(‘color’, ‘orange’)
```

And so on! This next method, `slideUp`, will cause a “slide” transition to occur on the selected elements. We’ll talk about transitions and animations in jQuery in the jQuery Effects lesson (see page 77). For now, just see how each method in the chain returns a jQuery object that can take additional methods.

```
$(‘h1’).text(‘hello!’).css(‘color’, ‘orange’).slideUp(1000).css(‘color’, ‘blue’).slide↓  
Down(1000);
```

You can chain together as many methods as you like and jQuery will run them in order from left to right.

When jQuery chains get too long, it can be difficult to read them all in a single line. For this reason, it’s common to format chains with each method after the first one on a separate line, like this:

```
$(‘h1’).text(‘hello!’)  
.css(‘color’, ‘orange’)  
.slideUp(1000)  
.css(‘color’, ‘blue’)  
.slideDown(1000);
```

Evaluation  
Copy

Open `Demos/jquery-function-and-selectors/chaining.html` in the browser to see this code in action. Notice that you never actually see the orange text. That is because the chained methods do not wait for animations to complete. They run one after another and complete in milliseconds. You will learn how to fix this in the jQuery Effects lesson (see page 81).

**EVALUATION COPY: Not to be used in class.**



## 2.7. Utility Functions

There are some jQuery methods, known as “core” methods that are called on `$` directly, rather than on the object returned by a selection.

Two commonly-used core methods are `$.trim()` and `$.each()`

### ❖ 2.7.1. `$.trim()`

The `$.trim()` method works the same as the JavaScript `trim()` method, which wasn't fully supported when jQuery was created. It removes all newlines, spaces, and tabs from the beginning and end of a string. To use it, pass a string to it as an argument, like this:

```
$.trim('      There are extra spaces.      ');
```

The result of running this statement will be the string "There are extra spaces." with the leading and trailing spaces removed.

View the following demo to see the `trim` method in action:

### Demo 2.3: Demos/jquery-function-and-selectors/trim.html

---

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.  <meta charset="UTF-8">
5.  <meta name="viewport" content="width=device-width, initial-scale=1">
6.  <title>jQuery's $.trim() Method</title>
7.  <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous"
8.         integrity="sha256-/xUj+3OJU5yExlq6GSYGSk7tPXikynS7ogEvDej/m4=></script>
9.  <script>
10.    $(function() {
11.      $('#trim-button').click(function() {
12.        $('#input2').val($.trim($('#input1').val()))
13.      })
14.    })
15.  </script>
16.  </head>
17.  <body>
18.    <h1>jQuery's $.trim() Method</h1>
19.    <p>Click the "Trim" button to trim the content of the first input and put it
20.      into the 2nd input.</p>
21.    <label for="input1">Input 1 (untrimmed):</label>
22.    <input type="text" id="input1" value="      words      " />
23.    <label for="input2">Input 2 (trimmed):</label>
24.    <input type="text" id="input2" />
25.    <button id="trim-button">Trim</button>
-----Lines 26 through 27 Omitted-----
```

---

## ❖ 2.7.2. `$.each()`

The `$.each()` method iterates over any object or array, passing each element to a callback function.

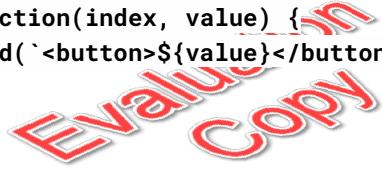
1. When used with an array, the index and value of each element are passed to the callback function.
2. When used with an object, the key and value of each property of the object are passed into the callback function.

For example, the following demo uses `$.each` to create a series of buttons from the elements in an array and appends them to an element with the id of “navBar”:

### Demo 2.4: Demos/jquery-function-and-selectors/each.html

---

```
-----Lines 1 through 9 Omitted-----  
10.  $(function() {  
11.    const navItems = ['home', 'about', 'contact'];  
12.  
13.    $.each(navItems, function(index, value) {  
14.      $('#navBar').append(`<button>${value}</button>`);  
15.    });  
16.  })  
17. </script>  
18. </head>  
19. <body>  
20. <h1>jQuery's $.each() Method</h1>  
21. <nav id="navBar"></nav>  
-----Lines 22 through 23 Omitted-----
```



---

### Code Explanation

This will produce the following result:

## jQuery's `$.each()` Method

home

about

contact

Note that the `$.each()` method is different from the `each()` method in jQuery, which only operates on jQuery objects.

## Conclusion

In this lesson, you have learned how to use selectors to create jQuery objects, how to chain jQuery methods together, how to use jQuery's utility methods, and how to cache jQuery objects.

# LESSON 3

## jQuery Manipulation

EVALUATION COPY: Not to be used in class.

### Topics Covered

- Adding elements.
- Removing elements.
- Replacing elements.
- CSS

### Introduction

DOM Manipulation is the process of using JavaScript to change what's displayed in a user's web browser. DOM manipulation methods can be used to get and set element content, get properties of elements, delete content and nodes from the DOM and more. In this lesson, you'll learn about the techniques and methods that jQuery uses to make the content and style of web pages more dynamic.

EVALUATION COPY: Not to be used in class.

\*

### 3.1. Getter and Setter Methods

Several of jQuery's DOM manipulation methods serve dual purposes. They can be used to retrieve current values from elements and to modify those values.

The full list of dual-purpose methods and their uses are:

1. `attr()` – gets or sets the value of an attribute.

```
$(‘div#someDiv’).attr(‘id’) // gets the id of the div  
$(‘div#someDiv’).attr(‘id’, ‘myDiv’) // sets the id of the div
```

2. `css()` – gets or sets CSS style properties.

```
$(‘div’).css(‘color’) // gets the CSS color property of the div  
$(‘div#myDiv’).css(‘color’, ‘#00F’) // sets the CSS color property of the div
```

3. `html()` – gets or sets the inner HTML of the matched elements.

```
$(‘div’).html() // gets the HTML of the div  
$(‘div#myDiv’).html(‘<strong>New Content!</strong>’) // sets the HTML of the div
```

4. `text()` – gets or sets the text content of the matched elements.

```
$(‘div’).text() // gets the text of the div  
$(‘div#myDiv’).text(‘some new text content’) // sets the text of the div
```

5. `val()` – gets the value of the first matched element, or sets the value of all of the matched elements.

```
$(‘input#firstName’).val() // gets the value of the input  
$(‘input#firstName’).val(‘new value here’) // sets the value of the input
```

6. `toggleClass()` – adds a class to an element if the element doesn’t have the class, and removes it if it does have it.

```
$(‘button#toggleButton’).click(function() {  
    $(this).toggleClass(“active”);  
})
```

In the preceding code, `this` is the button that was clicked. We pass it to `$()` to turn it into a jQuery object, so that we can call the `toggleClass()` method on it.

# Exercise 6: Getter and Setter Methods Practice

 10 to 15 minutes

This exercise will give you practice using these dual purpose methods.

From the `Exercises/manipulation` directory, open `getter-setter.html` in your browser and follow the instructions on the page to experiment with setting and getting DOM nodes.

The screenshot below shows the results you should expect:

```
> $('p').css('color');
<- "rgb(0, 0, 0)"

> $('p').css('color', '#ff0000');
<- ▶ k.fn.init(2) [p#introduction, p, prevObject: k.fn.init(1)]

> $('p').attr('id');
<- "introduction"

> $('#introduction').attr("id", "first-paragraph");
<- ▶ k.fn.init [p#first-paragraph, prevObject: k.fn.init(1)]

> $('p').attr('id');
<- "first-paragraph"

> $('h1').html('Fun with jQuery');
<- ▶ k.fn.init [h1, prevObject: k.fn.init(1)]

> $('p#first-paragraph').html();
<- "This demo is <em>your</em> chance to try out some
   of <strong>jQuery's DOM manipulation methods</strong>."

> $('p#first-paragraph').text();
<- "This demo is your chance to try out some
   of jQuery's DOM manipulation methods."
```

**EVALUATION COPY: Not to be used in class.**



## 3.2. Setting and Adding Content

In addition to the dual-purpose “getter/setter” methods, there are setter-only methods. These include:

1. `addClass()` – adds one or more classes to the matched elements.

```
$(‘div’).addClass(‘error’)
```

2. `before()` – inserts content before each of the matched elements.

```
$(‘div’).before(‘<div>new content</div>’)
```

A new `div` element will be added before each existing `div` element.

3. `after()` – inserts content after each of the matched elements.

```
$(‘div’).after(‘<div>new content</div>’)
```

A new `div` element will be added after each existing `div` element.

4. `insertBefore()` – the same as `before()` but with the source and target reversed.

```
$(‘<div>new content</div>’).insertBefore(‘div’)
```

This is the equivalent of:

```
$(‘div’).before(‘<div>new content</div>’)
```

5. `insertAfter()` – the same as `after()` but with the source and target reversed.

```
$(‘<div>new content</div>’).insertAfter(‘div’)
```

This is the equivalent of:

```
$(‘div’).after(‘<div>new content</div>’)
```

6. `prepend()` – inserts content to the beginning of each of the matched elements.

```
$(‘div’).prepend(‘<div>new content</div>’)
```

A new `div` element will be added as the first child of each existing `div` element.

7. `append()` – inserts content to the end of each of the matched elements.

```
$(‘div’).append(‘<div>new content</div>’)
```

A new `div` element will be added as the last child of each existing `div` element.

8. `prependTo()` – the same as `prepend()` but with the source and target reversed.

```
$( '<div>new content</div>' ).prependTo('div')
```

This is the equivalent of:

```
$( 'div' ).prepend('<div>new content</div>')
```

9. `appendTo()` – the same as `append()` but with the source and target reversed.

```
$( '<div>new content</div>' ).appendTo('div')
```

This is the equivalent of:

```
$( 'div' ).append('<div>new content</div>')
```

## ~~Evaluation Copy~~ before()/insertBefore() and after()/insertAfter()

An important difference between `before()` and `after()` and their reversed equivalents, `insertBefore()` and `insertAfter()`, is that the former two will return the previously existing element, while the latter two will return the new element. This is particularly important when chaining. Consider the following two pieces of code:

```
$( 'h1' ).after('<div>new content</div>').css('font-style', 'italic');
```

```
$( '<div>new content</div>' ).insertAfter('h1').css('font-style', 'italic');
```

1. Both lines of code will add a new `div` element after the existing `h1` element.
2. After the first line of code runs, the `h1` element will be italicized.
3. After the second line of code runs, the new `div` element will be italicized.

The same concept applies to `prepend()/prependTo()` and `append()/appendTo()`.

# Exercise 7: Setting and Adding Content

 10 to 15 minutes

This exercise will give you practice setting and adding content.

From the `Exercises/manipulation` directory, open `setter.html` in your browser and follow the instructions on the page to experiment with setting DOM nodes.

The screenshot below shows the results you should expect:

```
> $('li').append('Great Fun!');
< ▷ k.fn.init(9) [li, li, li, li, li, li, li, li, li, prevObject: k.fn.init(1)]
> $('p').after('<p>This is a new paragraph!</p>');
< ▷ k.fn.init(2) [p#introduction, p, prevObject: k.fn.init(1)]
> $('p').after('<p>This is a new paragraph!</p>');
< ▷ k.fn.init(4) [p#introduction, p, p, p, prevObject: k.fn.init(1)]
> $('ol').addClass('error warning danger');
< ▷ k.fn.init [ol.error.warning.danger, prevObject: k.fn.init(1)]
> $('ol').attr('class');
< "error warning danger"
> $('ol').addClass('cheese');
< ▷ k.fn.init [ol.error.warning.danger.cheese, prevObject: k.fn.init(1)]
> $('ol').attr('class');
< "error warning danger cheese"
```



**EVALUATION COPY: Not to be used in class.**



## 3.3. Copying and Removing Content

There are additional DOM manipulation methods for copying and removing content:

1. `clone()` – creates a copy of the matched elements.

```
$('#element').clone()
```

2. `remove()` – removes the matched elements from the DOM.

```
$('#element').remove()
```

3. `removeAttr()` – removes an attribute from each matched element.

```
| $('#element').removeAttr('title')
```

4. `replaceWith()` - replaces each matched element with provided new content.

```
| $('#element').replaceWith('<div>new content</div>')
```

5. `replaceAll()` – the same as `replaceWith()`, but with the source and target reversed.

```
| $('<div>new content</div>').replaceAll('#element')
```

6. `removeClass()` – removes a specific value from a `class` attribute.

```
| $('h2').removeClass('class1 class2')
```

# Exercise 8: Setting and Adding Content

 10 to 15 minutes

This exercise will give you practice copying and removing content.

From the `Exercises/manipulation` directory, open `remove-and-copy.html` in your browser and follow the instructions on the page to experiment with setting DOM nodes.

The screenshot below shows the results you should expect:

```
> $('h1').appendTo('p');
<- ► k.fn.init(2) [h1, h1, prevObject: k.fn.init(1)]
> $('h1:first').clone().prependTo('body');
<- ► k.fn.init [h1, prevObject: k.fn.init(1)]
> $('h1').replaceWith('<h2>The New H2</h2>');
<- ► k.fn.init(3) [h1, h1, h1, prevObject: k.fn.init(1)]
> $('<h3>New H3</h3>').replaceAll('h2');
<- ► k.fn.init(3) [h3, h3, h3, prevObject: k.fn.init(1)]
```

  
**EVALUATION COPY: Not to be used in class.**



## 3.4. event.target

When an event happens on a DOM element in JavaScript, an event object is created. This object is automatically passed into the callback function by the event listener. The event object contains properties that describe the event, and methods that can operate on the event. One of the most useful event properties is `event.target`, which returns the element that triggered the event. Knowing the element that triggered an event allows you to get its value.

For example, in jQuery you can listen for an event by using the `on` method. The `on` method listens for events that you pass as the first argument, and calls a function when that event happens. Here is an example of a simple use of the `on` method to listen for clicks on a button and print a message when the button is clicked:

```
$(‘button’).on(‘click’, function() {  
    console.log(“clicked”);  
})
```

If you want to print the value of the button that was clicked, you need to capture the event object and assign it to a variable. You do this by defining a parameter in the callback function’s parameter list. This can be anything, but it’s customary to name it “event” or just “e”. Once you have the event target, you can convert it into a jQuery object to make working with it and returning its value easier, as in this example:

```
$(‘button’).on(‘click’, function(event) {  
    const target = $(event.target);  
    console.log(target.text() + “ button was clicked”);  
});
```

You can also use `event.target` to find out which key was pressed when you listen for a `keyup` event. To do this, get the value of the `key` property of the event object:

```
$(‘body’).on(‘keyup’, function(event) {  
    console.log(event.key + “ was pressed”);  
});
```

Evaluation  
Copy

**EVALUATION COPY: Not to be used in class.**



## 3.5. Properties vs. Attributes

Properties and attributes of elements can both be used to access values of HTML element attributes, but there are cases where it’s preferable to get the property value. The method for accessing element properties is `prop()`. The most common use for the `prop()` method is to check whether a checkbox is checked. For example, consider the following HTML element:

```
<input id=“my-checkbox” type=“checkbox” checked=“checked”>
```

The `checked` attribute only sets the default checked status of this element. If you want to find out the current checked status of an element, you need to do so by getting the `checked` property, like this:

```
$( '#my-checkbox' ).prop( 'checked' );
```

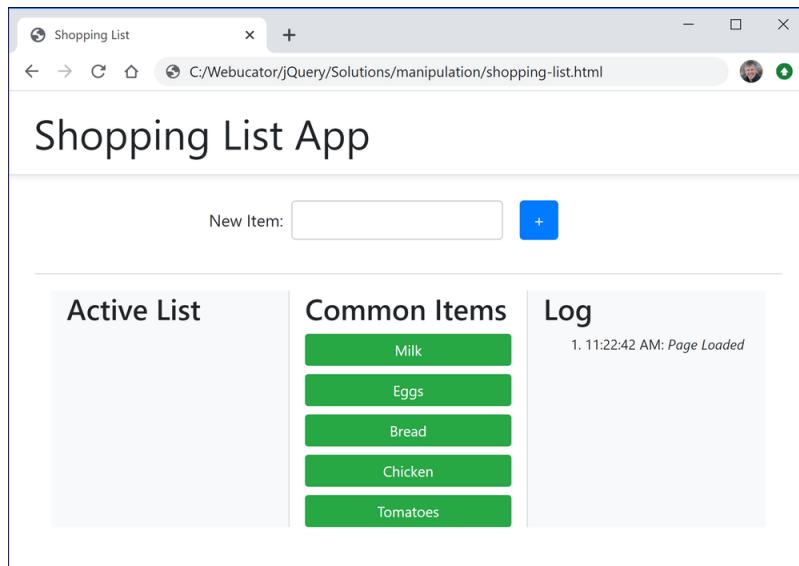
This will return either `true` or `false`, depending on whether the checkbox is currently checked.

**EVALUATION COPY: Not to be used in class.**

\*

## 3.6. Shopping List Application

Using what you have learned in this lesson, you will build the one-page shopping list application shown below:



Open `Solutions/manipulation/shopping-list.html` in your browser to see how the finished application works:

1. Notice that “Page Loaded” is logged and the **New Item** field gets focus.
2. Add Milk by clicking the **Milk** button under **Common Items**.
3. Add Lettuce by typing “Lettuce” in the **New Item** field and pressing the **+** button. Notice the **New Item** field gets focus, making it easy to enter another value.
4. Add Bread by typing “Bread” in the **New Item** field and pressing the **Enter** key.

5. Try adding Bread again using the **New Item** field. It should fail.
6. Try pressing the **+** button next to an empty **New Item** field. It should fail silently.
7. Try entering just spaces in the **New Item** field and pressing the **+** button. It should fail silently.
8. Remove Milk by clicking the **X** button next to Milk under **Active List**.

The HTML (`Exercises/manipulation/shopping-list.html`) has already been completed. You will build the JavaScript (`Exercises/manipulation/shopping-list.js`) piece by piece.



# Exercise 9: Logging

⌚ 15 to 25 minutes

In this exercise, you will complete the `log(msg)` function.

1. Open `Exercises/manipulation/shopping-list.html` in your editor. Examine the section of the code shown below. The ordered list will contain the log. You will need to access that ordered list and add list items to it with JavaScript.

```
<section id="log" class="col bg-light">
  <h3>Log</h3>
  <ol></ol>
</section>
```

2. Open `Exercises/manipulation/shopping-list.js` in your editor.
3. Notice that the `init()` function is called as the page loads, and the `init()` function passes “Page Loaded” to the `log()` function:

```
function init() {
  log('Page Loaded');
}

$(init);
```

4. In the `log(msg)` function, write jQuery code to:
  - A. Access the ordered list shown above and save it in a constant.
  - B. Create a new list item element and save it in a constant.
  - C. Get the current date and save it in a constant.
  - D. Set the `innerHTML` of the new list item to the current local time using the `toLocaleTimeString()` method, followed by a colon, followed by the `msg` passed to `log(msg)`. For example, “5:53:12 PM: <em>Page Loaded</em>”.
  - E. Append the new list item to the ordered list.
5. Test your code in the browser. When the page loads, it should log “Page Loaded”. If it isn’t working, use the console to help you debug.



## Solution: Solutions/manipulation/shopping-list-1.js

---

```
1.  /* Log Messages */
2.  function log(msg) {
3.    // Access the ordered list and save it in a variable
4.    const log = $('section#log>ol');
5.    // Create a new list item element and save it in a variable
6.    const newItem = $("<li></li>");
7.    // Get the current date and save it in a variable
8.    const now = new Date();
9.    // Set the innerHTML of the new list item
10.   newItem.html(now.toLocaleTimeString() + ': <em>' + msg + '</em>');
11.   // Append the new list item to the ordered list
12.   log.append(newItem);
13. }
14.
15. /* Remove item from list */
16. function removeFromList(e) {
17.   log('Item Removed');
18. }
19.
20. /* Add product to list */
21. function addToList(product) {
22.   log(product + ' added.');
23. }
24.
25. function init() {
26.   log('Page Loaded');
27. }
28.
29. $(init);
```

---



# Exercise 10: Adding EventListeners



25 to 40 minutes

In this exercise, you will add EventListeners in the `init()` function so that you can log when a new item is added. You will not yet write the code to actually add the items. You will do that in the next exercise.

1. Open `Exercises/manipulation/shopping-list.html` in your editor and notice the buttons and the `input` element. You will have to listen for the following events:
  - A. Clicks on any `button` element within the `section` with the `id` “common-items”.
  - B. Clicks on the `button` element with the `id` “add-new-item”.
  - C. KeyUp events on the `input` element with the `id` “new-item”.
2. Open `Exercises/manipulation/shopping-list.js` in your editor if it isn’t already open.
3. Beneath the `log('Page Loaded');` line, declare the following three constants:
  - A. `btnAddNewItem` - The `button` element with the `id` “add-new-item”.
  - B. `cnnItemsButtons` - The `button` elements within the `section` with the `id` “common-items”.
  - C. `newItem` - The `input` element with the `id` “new-item”.
4. Add a line of code to place focus on the `newItem` `input`, so the user can just start typing in a new item.
5. Each button in the `cnnItemsList` is coded as follows:

```
<button class="btn btn-success btn-sm btn-block btn-add" name="Milk">  
  Milk  
</button>
```

When the user clicks one of these buttons, your code should pass the name of that button as the argument for `product` to the `addToList(product)` function.

6. The `add-new-item` button is coded as follows:

```
<button id="add-new-item" class="btn btn-primary mx-2 my-2">+</button>
```

And the associated text field is:

```
<input class="form-control mx-2" id="new-item">
```

When the user clicks the “add-new-item” button, your code should:

- A. Pass the value of the text field as the argument for `product` to the `addToList(product)` function.
  - B. Clear the text field.
  - C. Place focus on the text field.
7. Finally, you need to add an `EventListener` for the `keyup` event on the “new-item” text field. The callback function should check if the key pressed was the **Enter** key. If it was, it should:
- A. Pass the value of the text field as the argument for `product` to the `addToList(product)` function.
  - B. Clear the text field.
  - C. Place focus on the text field.
8. Test your code in the browser. At this point, the shopping lists won’t change, but logging should work when you add new items. If it isn’t working, use the console to help you debug.



## Solution: Solutions/manipulation/shopping-list-2.js

---

```
-----Lines 1 through 19 Omitted-----  
20. function init() {  
21.     log('Page Loaded');  
22.     const btnAddNewItem = $('#add-new-item');  
23.     const cmnItemsButtons = $('#common-items>button');  
24.     const newItem = $('#new-item');  
25.     newItem.focus();  
26.  
27.     /* Add event listeners to all common list Add buttons */  
28.     cmnItemsButtons.on('click', function() {  
29.         addToList($(this).attr('name'));  
30.         newItem.focus();  
31.     })  
32.  
33.     /* Add event listener to New Item Add button */  
34.     btnAddNewItem.on('click', function() {  
35.         addToList(newItem.val());  
36.         newItem.val('');  
37.         newItem.focus();  
38.     });  
39.  
40.     /*  
41.      Add event listener capturing Enter press while  
42.      focus is on New Item field  
43.    */  
44.     newItem.on('keyup', function(e) {  
45.         if (e.key === 'Enter') {  
46.             addToList(newItem.val());  
47.             newItem.val('');  
48.             newItem.focus();  
49.         }  
50.     });  
51. }  
52.  
53. $(init);
```

---

Evaluation  
Copy

# Exercise 11: Adding Items to the List

 15 to 25 minutes

---

In this exercise, you will write the `addToList()` function.

1. Open `Exercises/manipulation/shopping-list.js` in your editor if it isn't already open.
2. Currently, the `addToList()` function should look like this:

```
function addToList(product) {  
  log(product + ' added.')  
}
```

Above the `log(product + ' added.')` line, you will write code that does the following:

- A. Removes leading and trailing whitespace from the passed-in product, so that if the user enters “ Milk ”, we store it as “Milk”.
  - B. Accesses the “active-items-list” unordered list and saves it in a constant.
  - C. Creates a new list item element and saves it in a constant.
  - D. Sets the title of the new list item element to the product name.
  - E. Sets the `innerHTML` of the new list item element to the product name.
  - F. Adds these Bootstrap classes to the new list item element:  
  
`list-group-item d-flex justify-content-between align-items-center py-1`
  - G. Appends the new list item to the “active-items-list” unordered list.
3. Test your code in the browser. You should now be able to add items to list. If it isn't working, use the console to help you debug.

## Solution: Solutions/manipulation/shopping-list-3.js

---

```
-----Lines 1 through 15 Omitted-----
16. function addToList(product) {
17.     product = $.trim(product);
18.
19.     const activeList = $('#active-items-list');
20.     const newItem = $('- </li>');
21.     newItem.attr("title", product);
22.     newItem.html(product);
23.     newItem.addClass(
24.         'list-group-item d-flex justify-content-between align-items-center py-1'
25.     );
26.     activeList.append(newItem);
27.     log(product + ' added.');
28. }
-----Lines 29 through 63 Omitted-----

```

---

# Exercise 12: Dynamically Adding Remove Buttons to the List Items

 15 to 25 minutes

In this exercise, you will continue to work in the `addToList()` function. You will add “remove” buttons to the list items you created in the last exercise.

1. Open `Exercises/manipulation/shopping-list.js` in your editor if it isn’t already open.
2. Currently, the `addToList()` function should look something like this:

```
function addToList(product) {  
    product = $.trim(product);  
  
    const activeList = $('#active-items-list');  
    const newItem = $('- </li>');  
    newItem.attr("title", product);  
    newItem.html(product);  
    newItem.addClass(  
        'list-group-item d-flex justify-content-between align-items-center py-1'  
    );  
    activeList.append(newItem);  
    log(product + ' added.');  
}

```

You will write code below the `log(product + ' added.')` line that does the following:

- A. Creates a button element with a minus sign that calls `removeFromList()` when clicked and appends the button to the new list item.
- B. Checks if the list item being added is one of the common item buttons. If it is, the code disables the button by setting its `disabled` property to `true`. **Hint:** Look at the name attributes of the buttons in the “common-items” section. Can you find a button with the same name as the new list item you’re adding?

Note that these directions are intentionally less specific than in the previous exercises.

3. Test your code in the browser. The list items in the “active-items-list” list should now have remove buttons. They won’t actually remove the items, but they should log “Item removed” when clicked. Also, any item in the “common-items” section that is also in the “active-items-list” should be disabled. If your code isn’t working, use the console to help you debug.

## Solution: Solutions/manipulation/shopping-list-4.js

---

```
-----Lines 1 through 15 Omitted-----
16. function addToList(product) {
17.     product = $.trim(product);
18.
19.     const activeList = $('#active-items-list');
20.     const newItem = $('- </li>');
21.     newItem.attr("title", product);
22.     newItem.html(product);
23.     newItem.addClass(
24.         'list-group-item d-flex justify-content-between align-items-center py-1'
25.     );
26.     activeList.append(newItem);
27.     log(product + ' added.');
28.
29.     const btnRemove = $('<button class="btn btn-sm btn-danger">X</button>');
30.     btnRemove.on('click', removeFromList);
31.     newItem.append(btnRemove);
32.
33.     // Check if list item being added is in common list items
34.     // If it is, we need to disable its button there.
35.     const selector = '#common-items>button[name="' + product + '"]';
36.     const btnMatch = $(selector);
37.     if (btnMatch) {
38.         btnMatch.prop('disabled', true);
39.     }
40. }

-----Lines 41 through 75 Omitted-----

```

# Exercise 13: Removing List Items

 15 to 25 minutes

---

In this exercise, you will write the `removeFromList()` function to remove elements from the “active-items-list” list.

1. Open `Exercises/manipulation/shopping-list.js` in your editor if it isn’t already open.
2. Currently, the `removeFromList()` function should look like this:

```
function removeFromList(e) {  
  log('Item Removed');  
}
```

- 
- A. Using the passed-in event (`e`), access the list item that contains the button that was clicked to call this function and assign that list item to a constant.
  - B. Remove that item from the list.
  - C. Change `log('Item Removed')` to log the name of the product removed (using its `title` attribute).
  - D. Check if the list item being removed is in the common list. If it is, re-enable the associated button by setting its `disabled` property to `false`.
3. Test your code in the browser. When a remove button is clicked, the associated list item should now get removed and the log should tell you which item was removed. In addition, if there is an associated list item in the “common-list-items” list, that button should be re-enabled. If your code isn’t working, use the console to help you debug.

## Solution: Solutions/manipulation/shopping-list-5.js

---

```
-----Lines 1 through 9 Omitted-----  
10. /* Remove item from list */  
11. function removeFromList(e) {  
12.   const item = $(e.target).parent();  
13.   item.remove();  
14.   const title = item.attr('title');  
15.   log(title + ' removed.')  
16.  
17.   const selector = '#common-items>button[name="${title}"]';  
18.   console.log(selector);  
19.   const btnMatch = $(selector);  
20.   if (btnMatch) {  
21.     btnMatch.prop('disabled', false);  
22.   }  
23. }  
-----Lines 24 through 85 Omitted-----
```

---

# Exercise 14: Preventing Duplicates and Zero-length Product Names

 15 to 25 minutes

In this exercise, you will finalize the shopping list by preventing duplicate values and empty strings from being added to the “active-items-list” list.

1. There are a couple of issues still. Open `Exercises/manipulation/shopping-list.html` in your browser.
2. Add Milk via the **Common Items** list and then try adding it again using the **New Item** form field. Milk will be listed twice in your **Active List**. We’ll fix that.
3. Press the **+** button next to the empty **New Item** form field. It will add an empty item to your **Active List**. We’ll fix that too.
4. Open `Exercises/manipulation/shopping-list.js` in your editor if it isn’t already open.
5. In the `addToList()` function, below the line in which you trim the product name, add code that checks if the trimmed product name is an empty string, and if it is, return `false` so that the rest of the code in the function doesn’t run.
6. Below the code you just wrote, check if the passed-in product is already listed in the “active-items-list” list. If it is, log a message saying something like “Not adding Chicken again.” and return `false`.
7. Test your code in the browser.
  - A. Add Milk via the **Common Items** list and then try adding it again using the **New Item** form field. It should fail and log that it’s not adding Milk again.
  - B. Press the **+** button next to the empty **New Item** form field. It should fail silently.
8. If your code isn’t working, use the console to help you debug.

## Solution: Solutions/manipulation/shopping-list.js

---

```
-----Lines 1 through 24 Omitted-----
25. /* Add product to list */
26. function addToList(product) {
27.     product = $.trim(product);
28.     if (!product.length) {
29.         return false; // nothing to add
30.     }
31.
32.     // Check if list item is already in active list
33.     const liMatch = $('#active-items-list>li[title="' + product + '"]');
34.     if (liMatch.length) {
35.         log("Not adding " + product + " again.");
36.         return false;
37.     }
38.
39.     const activeList = $('#active-items-list');
40.     const newItem = $('- 
');
41.     newItem.attr("title", product);
42.     newItem.html(product);
43.     newItem.addClass(
44.         'list-group-item d-flex justify-content-between align-items-center py-1'
45.     );
46.     activeList.append(newItem);
47.     log(product + ' added.');
48.
49.     const btnRemove = $('<button class="btn btn-sm btn-danger">X</button>');
50.     btnRemove.on('click', removeFromList);
51.     newItem.append(btnRemove);
52.
53.     // Check if list item being added is in common list items
54.     // If it is, we need to disable its button there.
55.     const selector = '#common-items>button[name="' + product + '"]';
56.     const btnMatch = $(selector);
57.     if (btnMatch) {
58.         btnMatch.prop('disabled', true);
59.     }
60. }

-----Lines 61 through 95 Omitted-----
```

---

# Conclusion

In this lesson, you have learned how to use jQuery's DOM manipulation methods to get, set, copy, and remove nodes from the browser DOM.

Evaluation  
Copy



# LESSON 4

## jQuery Forms and Events

EVALUATION COPY: Not to be used in class.

### Topics Covered

- Listening for events.
- Responding to events.
- Triggering events.
- `on()`, `off()`, and `trigger()`.
- Event delegation.

### Introduction

Web browsers emit events in response to many different actions happening inside an HTML document, such as the click, mouseover, load, and change events. In this lesson, you'll learn about how to listen for and respond to events using jQuery.

Evaluation Copy

EVALUATION COPY: Not to be used in class.

\*

### 4.1. Listening for Events

An `EventListener` represents an object that does something when an event occurs. Think of a swimmer on a block, waiting for the starting gun to go off. When the gun goes off, the swimmer dives. Here is some pseudo-code to set that up in JavaScript:

```
diver.addEventListener('shotFire', dive);
```

In the pseudo-code above, `diver` is the `EventTarget`, `shotFire` is the event type, and `dive` is the function that will be called when the event occurs. Functions that are called in response to an event are known as *callback functions*.

An `EventTarget` is any object on which an event can occur, including `window`, `document`, and any HTML element. The basic syntax is as follows:

```
object.addEventListener(eventType, callbackFunction);
```

Some of the most common event types are:

1. `blur` – The element lost the focus.
2. `change` – The element value was changed.
3. `click` – A pointer button was clicked.
4. `dblclick` – A pointer button was double-clicked.
5. `focus` – The element received the focus.
6. `keydown` – A key was pressed down.
7. `keyup` – A key was released.
8. `mousedown` – A pointer button was pressed down.
9. `mousemove` – A pointer was moved within the element.
10. `mouseout` – A pointer was moved off of the element.
11. `mouseover` – A pointer was moved onto the element.
12. `mouseup` – A pointer button was released over the element.
13. `reset` – The form was reset.
14. `select` – Some text was selected.
15. `submit` – The form was submitted.

### ❖ 4.1.1. jQuery's `on()` Method

As discussed, in vanilla JavaScript, you listen for these events using the `addEventListener()` method. The jQuery equivalent is the `on()` method.

### Vanilla JavaScript

```
document.getElementById('my-button').addEventListener('click', (e) => {
    alert('You clicked the button!');
})
```

### jQuery

```
$('#my-button').on('click', (e) => {
    alert('You clicked the button!');
})
```

You can also use the `on()` method to attach multiple event handlers to an element at once, by passing in an object containing event names and functions. In the following demo, the same button will have CSS styles applied when the mouse enters, leaves, or clicks the button:

## Demo 4.1: Demos/forms-and-events/on-button.html

---

```
-----Lines 1 through 8 Omitted-----
9.  <script>
10.  $(function() {
11.      $('button').on({
12.          click: (e) => {
13.              $(e.target).css('background-color', 'pink');
14.          },
15.          mouseover: (e) => {
16.              $(e.target).css('background-color', 'yellow');
17.          },
18.         mouseout: (e) => {
19.              $(e.target).css('background-color', 'aqua');
20.          }
21.      });
22.  });
23. </script>
-----Lines 24 through 29 Omitted-----
```

---

### Code Explanation

---

Open the file in the browser, hover on, click, and hover off the button to see that the button is listening for all of these events.

---

## ❖ 4.1.2. Removing EventListeners

Sometimes, it's necessary to remove EventListeners. In vanilla JavaScript, this is done with the `removeEventListener()` method, which takes two arguments: the event and the callback function:

```
object.removeEventListener(eventType, callbackFunction);
```

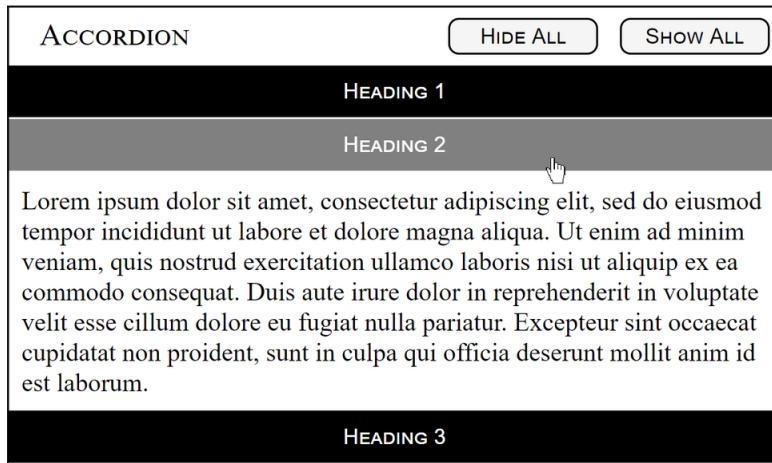
In jQuery, you remove EventListeners with the `off()` method. There are several ways of using `off()`:

1. `object.off(eventType, callbackFunction)` – Stop calling this callback function for this event type on this object.
2. `object.off(eventType)` – Stop listening for this event type on this object.
3. `object.off()` – Stop listening for all event types on this object.

## ❖ 4.1.3. Shorthand Event Handler Methods

jQuery provides shorthand methods that can be used as an alternative to the `on()` method. These methods use the name of the event. For example, `on('click', callbackFunction)` can be written as `click(callbackFunction)`.

The following demo shows how to create this accordion with vanilla JavaScript:



Before looking at the code, open `Demos/forms-and-events/accordion-vanilla.html` in your browser.

**Things to notice:**

1. When you hover over a heading, the paragraph that follows it becomes visible.
2. When you hover off the heading, the paragraph that follows it disappears.
3. When you click a heading, the paragraph that follows it stays open. The button is no longer listening for `mouseout` events to hide the paragraph.
4. When you click the **Show All** button, all paragraphs are shown and locked open. None of the buttons are listening for `mouseout` events that hide paragraphs.
5. When you click the **Hide All** button, all paragraphs are hidden and all event listening is turned back on.

## Demo 4.2: Demos/forms-and-events/accordion-vanilla.html

---

```
-----Lines 1 through 6 Omitted-----
7. <script>
8.     function open(e) {
9.         btn = e.currentTarget;
10.        btn.nextElementSibling.style.display = 'block';
11.    }
12.
13.    function close(e) {
14.        btn = e.currentTarget;
15.        btn.nextElementSibling.style.display = 'none';
16.    }
17.
18.    window.addEventListener('load', function() { // listener added
19.        const accordionButtons = document.querySelectorAll('.accordion>button');
20.        accordionButtons.forEach((button) => {
21.            button.addEventListener('click', (e) => { // listener added
22.                const thisButton = e.currentTarget;
23.                thisButton.removeEventListener('mouseout', close); // listener removed
24.                thisButton.classList.add('locked');
25.            });
26.
27.            button.addEventListener('mouseover', open); // listener added
28.            button.addEventListener('mouseout', close); // listener added
29.        });
30.
31.        const showAllButtons = document.querySelectorAll(
32.            '.accordion>heading>.show-all'
33.        );
34.        const hideAllButtons = document.querySelectorAll(
35.            '.accordion>heading>.hide-all'
36.        );
37.
38.        showAllButtons.forEach((button) => {
39.            button.addEventListener('click', (e) => { // listener added
40.                const accordion = e.currentTarget.parentNode.parentNode;
41.                const ps = accordion.querySelectorAll('p');
42.                ps.forEach((p) => {
43.                    p.style.display = 'block';
44.                });
45.                const btns = accordion.querySelectorAll(':scope>button');
46.                btns.forEach((btn) => {
47.                    btn.removeEventListener('mouseout', close); // listener removed
48.                    btn.classList.add('locked');
49.                });
50.            });
51.        });
52.    });
53.
```

```
50.      });
51.    });
52.
53.    hideAllButtons.forEach((button) => {
54.      button.addEventListener('click', (e) => { // listener added
55.        const accordion = e.currentTarget.parentNode.parentNode;
56.        const ps = accordion.querySelectorAll('p');
57.        ps.forEach((p) => {
58.          p.style.display = 'none';
59.        });
60.        const btns = accordion.querySelectorAll(':scope>button');
61.        btns.forEach((btn) => {
62.          btn.addEventListener('mouseout', close); // listener added
63.          btn.classList.remove('locked');
64.        });
65.      });
66.    });
67.  });
68. </script>
```

-----Lines 69 through 109 Omitted-----

---

Evaluation  
Copy

## Code Explanation

---

Spend some time reviewing this code. Notice each place where an `EventListener` is added or removed.

---

Here is the same file rewritten in jQuery:

## Demo 4.3: Demos/forms-and-events/accordion-jquery.html

---

```
-----Lines 1 through 8 Omitted-----
9. <script>
10. $(function() { // listener added
11.     const accordionButtons = $('.accordion>button');
12.     accordionButtons.on({
13.         click: (e) => { // listener added
14.             const thisButton = $(e.target);
15.             thisButton.off('mouseout'); // listener removed
16.             thisButton.addClass('locked');
17.         },
18.         mouseover: (e) => { // listener added
19.             $(e.target).next().show();
20.         },
21.         mouseout: (e) => { // listener added
22.             $(e.target).next().hide();
23.         }
24.     });
25.
26.     const showAllButtons = $('.accordion>heading>.show-all');
27.     const hideAllButtons = $('.accordion>heading>.hide-all');
28.
29.     showAllButtons.click((e) => { // listener added
30.         const accordion = $(e.target).parents('.accordion');
31.         accordion.children('p').show();
32.         accordion.children('button').off('mouseout'); // listener removed
33.         accordion.children('button').addClass('locked');
34.     });
35.
36.     hideAllButtons.click((e) => { // listener added
37.         const accordion = $(e.target).parents('.accordion');
38.         accordion.children('p').hide();
39.         accordion.children('button').mouseout((e) => { // listener added
40.             $(e.target).next().hide();
41.         });
42.         accordion.children('button').removeClass('locked');
43.     });
44. });
45. </script>
-----Lines 46 through 86 Omitted-----
```

## Code Explanation

Again, spend some time reviewing this code, paying particular attention to where EventListeners are added and removed. Notice that, even with modern-day JavaScript, the jQuery method saves us some coding.

**EVALUATION COPY: Not to be used in class.**

\*

## 4.2. Triggering Events

Like many of jQuery's methods, the event shorthand methods are dual-purpose. They can be used both to detect events and to trigger events. For example, the following code will place the cursor in the "search" input:

```
$('input#search').focus();
```

Evaluation  
Copy

This is shorthand code for:

```
$('input#search').trigger('focus');
```

This is directly analogous to vanilla JavaScript's `focus()` method:

```
document.getElementById('search').focus();
```

**EVALUATION COPY: Not to be used in class.**

\*

## 4.3. Delegating Events

By using a technique called event delegation, you can attach one event listener to a parent element and use it to detect events on all of the elements inside of that element. Using this technique frees you from having to add event handling to new child elements that get added.

For example, review the following demo:

## Demo 4.4: Demos/forms-and-events/to-do-list.html

---

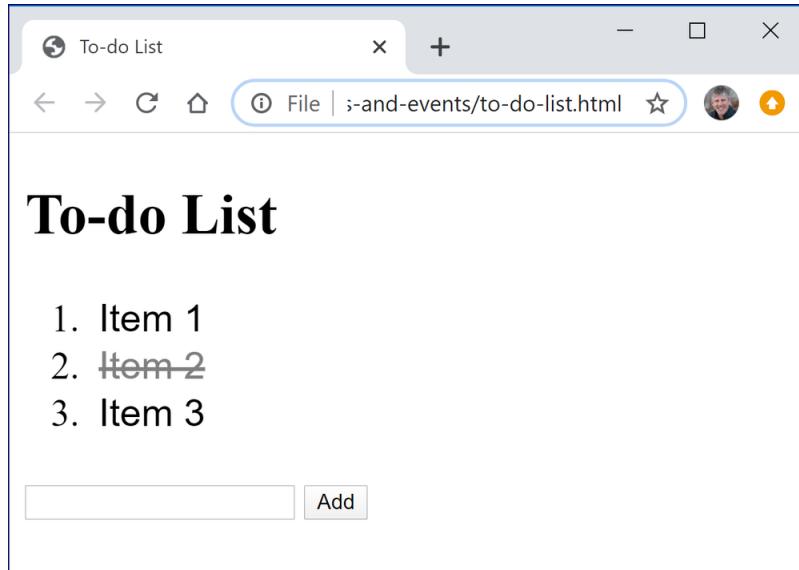
```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.  <meta charset="UTF-8">
5.  <meta name="viewport" content="width=device-width, initial-scale=1">
6.  <link rel="stylesheet" href="../../webucator.css">
7.  <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous"
8.      integrity="sha256-/xJj+30JU5yExlq6GSYGSK7tPXikynS7ogEvDej/m4=></script>
9.  <script>
10.    $(function() {
11.      const toDoList = $('#to-do-list');
12.      toDoList.find('li>button').click( (e) => {
13.        $(e.target).toggleClass('done');
14.      });
15.
16.      $('#add-item').click( (e) => {
17.        const item = $('#new-item');
18.        toDoList.append(`<li><button>${item.val()}</button></li>`);
19.        item.val('');
20.      })
21.    });
22.  </script>
23.  <title>To-do List</title>
24.  </head>
25.  <body id="to-do">
26.  <main>
27.    <h2>To-do List</h2>
28.    <ol id="to-do-list">
29.      <li><button>Item 1</button></li>
30.      <li><button>Item 2</button></li>
31.      <li><button>Item 3</button></li>
32.    </ol>
33.    <input id="new-item">
34.    <button id="add-item">Add</button>
35.  </main>
36.  </body>
37. </html>
```

Evaluation  
Copy

---

### Code Explanation

1. Open `to-do-list.html` in your browser.
2. Click a list item. Notice it turns gray and gets crossed out:



3. Click the same list item again. It goes back to its original style.
4. Add a new item via the text box at the bottom. It should be added to the list.
5. Click the item you just added. Notice its style does not change. This is because the event listeners were added to the specific list items before the new list item was added:

```
const ToDoList = $('#to-do-list');
ToDoList.find('li>button').click( (e) => {
    $(e.target).toggleClass('done');
});
```

---

Now, consider this demo:

## Demo 4.5: Demos/forms-and-events/to-do-list-delegation.html

---

```
-----Lines 1 through 8 Omitted-----
9. <script>
10. $(function() {
11.   const toDoList = $('#to-do-list');
12.   toDoList.click( (e) => {
13.     $(e.target).toggleClass('done');
14.   });
15.
16.   $('#add-item').click( (e) => {
17.     const item = $('#new-item');
18.     toDoList.append(`<li><button>${item.val()}</button></li>`);
19.     item.val('');
20.   })
21. });
22. </script>
-----Lines 23 through 37 Omitted-----
```

### Code Explanation

---

- 
1. Open `to-do-list-delegation.html` in your browser.
  2. Add a new item via the text box at the bottom. It should be added to the list.
  3. Click the item you just added, and notice that its style changes. The difference is that we are listening for clicks on the whole list instead of the individual list items. We can do this, because `$(e.target)` returns the lowest level element that was clicked.

```
const toDoList = $('#to-do-list');
toDoList.click( (e) => {
  $(e.target).toggleClass('done');
});
```

---

The equivalent of `$e.target` in vanilla JavaScript is `e.target`. The equivalent vanilla JavaScript code is shown below:

## Demo 4.6: Demos/forms-and-events/to-do-list-delegation-vanilla.html

---

```
-----Lines 1 through 6 Omitted-----
7. <script>
8.   window.addEventListener('load', () => {
9.     const todoList = document.getElementById('to-do-list');
10.    todoList.addEventListener('click', (e) => {
11.      e.target.classList.toggle('done');
12.    });
13.
14.    document.getElementById('add-item').addEventListener('click', (e) => {
15.      const item = document.getElementById('new-item');
16.      const newItem = document.createElement('li');
17.      const newButton = document.createElement('button');
18.      newButton.innerHTML = item.value;
19.      newItem.appendChild(newButton);
20.      todoList.appendChild(newItem);
21.      item.value = '';
22.    });
23.  });
24. </script>
-----Lines 25 through 39 Omitted-----
```

---

# Exercise 15: Event Delegation

 10 to 15 minutes

---

In this exercise, you will fix an app that attempts to move items from one list to another.

1. Open `Exercises/forms-and-events/switch-lists.html` in your browser.
2. There are some fish in the birds list and birds in the fish list. Try to categorize these correctly by clicking items to move them from one list to the other.
3. Everything may appear to be working great. But see what happens when you make a mistake. Move a bird from the birds list to the fish list and then try to move it back. Notice that it doesn't work.
4. Open `Exercises/forms-and-events/switch-lists.html` in your editor and fix the code.

## Solution: Solutions/forms-and-events/switch-lists.html

---

```
-----Lines 1 through 8 Omitted-----  
9. <script>  
10. $(function() {  
11.     const list1 = $('#birds');  
12.     const list2 = $('#fish');  
13.  
14.     list1.click((e) => {  
15.         const li = $(e.target).parent();  
16.         li.appendTo(list2);  
17.     });  
18.  
19.     list2.click((e) => {  
20.         const li = $(e.target).parent();  
21.         li.appendTo(list1);  
22.     });  
23. };  
24. </script>  
-----Lines 25 through 61 Omitted-----
```

Evaluation  
Copy

## Conclusion

In this lesson, you have learned how to attach and remove event listeners to elements, how to trigger events programmatically, and how to use event delegation.

# LESSON 5

## jQuery Effects

**EVALUATION COPY: Not to be used in class.**

### Topics Covered

- Display effects.
- Fading effects.
- Sliding effects.
- Animating.

### Introduction

The visual effects that are built into jQuery, when used judiciously and creatively, can add a little extra spice to your web applications. Overuse of certain jQuery effects has led to them becoming somewhat synonymous with tacky and bad web design. In this lesson, you'll learn how to use jQuery's effects. How you decide to use them is up to you (but, hint, no one really wants elements that slide all over the page).

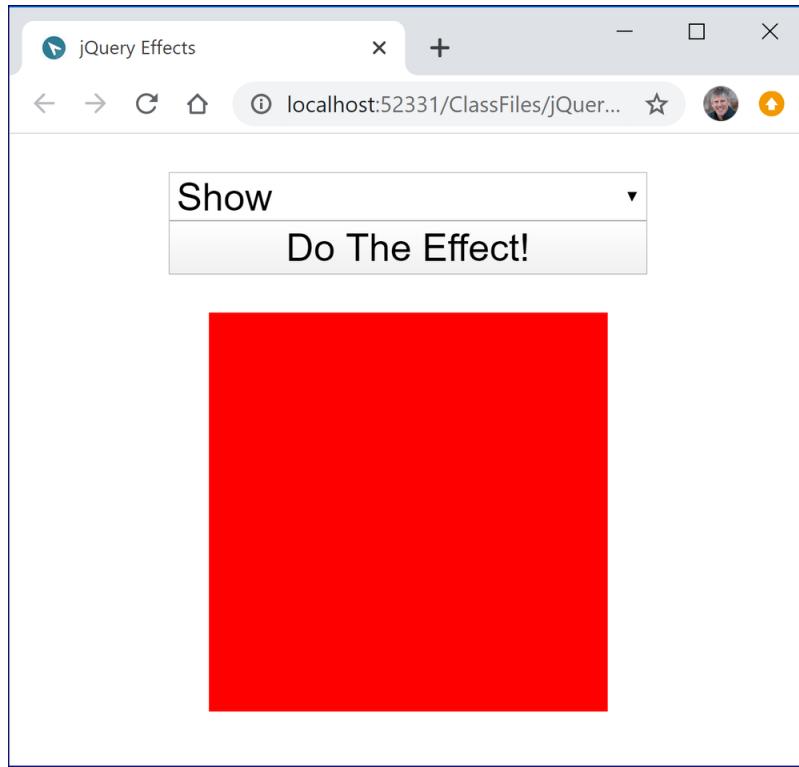
**EVALUATION COPY: Not to be used in class.**

\*

### 5.1. Display Effects

The display effects simply show and hide HTML elements. And, in fact, `show()` and `hide()` are the two main methods in this category of effects. The third one, `toggle()`, can be used to flip whether an element is currently showing or hiding to its opposite state.

To see each of the effects covered in this lesson in action, open `Demos/effects/effects.html` in your browser. Select an effect from the dropdown menu and then click the button. The selected effect will be run on the red square below the button:



**Note:** Effects that show or hide the element will have no effect on the red square if it's already shown or hidden. For example, if the element is showing and you select "Fade In," nothing will happen. However, if you select "Fade Out" while the square is showing, the element will fade until it disappears.

**EVALUATION COPY: Not to be used in class.**

\*

## 5.2. Fading Effects

Fading effects provide a simple way to control the opacity of an element over time. The fading effect methods are: `fadein()`, `fadeout()`, `fadetoggle()`, and `fadetoggle()`.

### ❖ 5.2.1. Fade method parameters

You can call the `fade` method without passing it any parameters, and the default values will be used. However, for more control over how the fade happens, you can pass in any combination of the following optional parameters:

- `duration` - controls how fast the fade effect will happen, in milliseconds.
- `easing` - specifies a “curve” to use for the speed of the effect. The default value is “swing,” which fades the element slower at the beginning and end than in the middle. The other possible value for the easing parameter is “linear,” which changes the effect at the same rate from beginning to end.
- `complete` - allows you to specify a function to run once the effect is complete.

Both the fade effects and the slide effects have a toggle version, which will do the opposite of what the current element state is. For example, if you run `fadetoggle()` on a visible element, it will cause it to fade out. If you run it on an invisible element, it will cause it to fade in.

### ❖ 5.2.2. Easing

Easing functions tell jQuery’s animations how to change the rate of animation over its duration. JQuery has two built-in easing functions, `swing` and `linear` through the use of jQuery plugins, you can make use of additional easing functions. The purpose of easing functions is to make animations more interesting or more “life-like.” In the following demo, you will see the difference between `swing` and `linear`:

## Demo 5.1: Demos/effects/easing.html

---

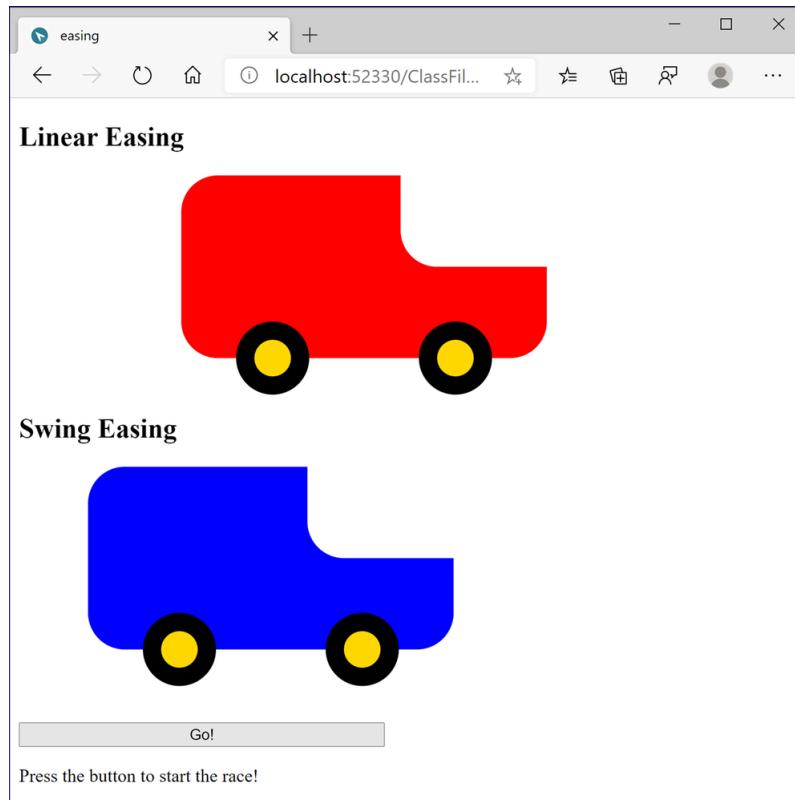
```
-----Lines 1 through 9 Omitted-----
10. <script>
11.   $(function() {
12.     $('#go').click(function() {
13.       $('#car1').animate({
14.         left: '+=800'
15.       }, 10000, 'linear');
16.       $('#car2').animate({
17.         left: '+=800'
18.       }, 10000, 'swing');
19.     });
20.   });
21. </script>
22. </head>
23. <body id="easing">
24.   <div>
25.     <h2>Linear Easing</h2>
26.     <div id="car1">
-----Lines 27 through 34 Omitted-----
35.   </div>
36.   <div>
37.     <h2>Swing Easing</h2>
38.     <div id="car2">
-----Lines 39 through 46 Omitted-----
47.   </div>
48.   <button id="go" class="btn btn-primary">Go!</button>
49.   <p>Press the button to start the race!</p>
-----Lines 50 through 51 Omitted-----
```

Evaluation  
Copy

---

### Code Explanation

Open the file in your browser to see the difference between `swing` and `linear` easing:



### ❖ 5.2.3. The complete Callback

Jquery's effect methods can take an optional function, called the **complete** callback, as a parameter. This function will run when the effect or animation is finished. The following demo uses the **complete** callback to display a message at the end of the animations from the previous demo:

## Demo 5.2: Demos/effects/easing-complete.html

---

```
-----Lines 1 through 9 Omitted-----
10. <script>
11.   $(function() {
12.     $("#go").click(function() {
13.       $("#car1").animate({
14.         left: "+=800"
15.       }, 10000, "linear", function() {
16.         $('#car1').css('background-color', 'silver');
17.       });
18.       $("#car2").animate({
19.         left: "+=800"
20.       }, 10000, "swing", function() {
21.         $('#car2').css('background-color', 'silver');
22.       });
23.     });
24.   });
25. </script>
-----Lines 26 through 55 Omitted
```

Evaluation  
Copy

---

### Code Explanation

---

Open the file in your browser to see the cars change color when the animation is complete.

---

#### ❖ 5.2.4. Revisiting the Accordion

The `show()` and `hide()` methods also can take easing functions. For example:

```
$(e.target).next().show(500, 'linear');
```

Open `Demos/effects/accordion.html` in your browser to see how the easing effects make the accordion much nicer. Open the file in your editor to review the code.

# Exercise 16: Waiting for Fading to Finish

 5 to 10 minutes

---

When we covered chaining (see page 29), we showed the following demo:

## **Exercise Code 16.1: Demos/jquery-function-and-selectors/chaining.html**

```
-----Lines 1 through 8 Omitted-----  
9.  <script>  
10.    $(function() {  
11.      $("h1").text("hello!")  
12.      .css("color", "orange")  
13.      .slideUp(1000)  
14.      .css("color", "blue")  
15.      .slideDown(1000);  
16.    });  
17.  </script>  
-----Lines 18 through 23 Omitted-----
```

Evaluation  
Copy

We explained that the user will never see the text of the h1 element turn orange, because the chained methods do not wait for animations to complete. You now know how to fix that.

1. From the Exercises/effects folder, open chaining.html for editing.
2. Modify the code so that the text of the h1 element is orange while it is fading out and blue while it is fading in.

## Solution: Solutions/effects/chaining.html

---

```
-----Lines 1 through 8 Omitted-----  
9. <script>  
10. $(function() {  
11.     $('h1').text('hello!')  
12.         .css('color', 'orange')  
13.         .slideUp(1000, function() {  
14.             $(this).css('color', 'blue');  
15.         })  
16.         .slideDown(1000);  
17.     });  
18. </script>  
-----Lines 19 through 24 Omitted-----
```

---

EVALUATION COPY: Not to be used in class.

### 5.3. Sliding Effects

~~Evaluation  
Copy~~

Sliding effects transition elements from hidden to shown or vice-versa by using a gradual movement effect. The slide effect methods are: `slidedown()`, `slideup()`, and `slidetoggle()`.

#### ❖ 5.3.1. Sliding effect parameters

Like the fade methods, the slide methods can also be called without parameters. However, they can also take the following optional parameters:

- `speed` - controls the speed of the effect, in milliseconds.
- `callback` - allows you to specify a function to run once the effect is complete.

EVALUATION COPY: Not to be used in class.

\*

## 5.4. Animating CSS Properties

The `jQuery animate()` method performs custom animations by changing numeric CSS properties over time.

The `animate()` method takes a JavaScript object as its first parameter, which defines the style properties that the element should be animated to. For example, in the following code, the selected element will change its background color to green. If you don't specify a speed, this animation will take 400 milliseconds.

```
$('#my-div').animate({backgroundColor: '#00ff00'});
```

The second parameter of `animate` is the speed of the animation. Like the `fade` and `slide` methods, the possible values of the speed parameter are “slow”, “fast”, or a number of milliseconds.

Using the `animate()` method, it is possible to animate any number of style properties simultaneously, as in the following example:

```
$('#my-div').animate({
  backgroundColor: '#ff00ff',
  width: "200px",
  height: "300px"
})
```

You can also change the position of an element using `animate`. However, to do so, you must first set the `position` property of the element to “relative”, “fixed”, or “absolute.”

## Conclusion

In this lesson, you have learned how to use jQuery's effect methods to simplify the task of creating JavaScript animations and transitions.



# LESSON 6

## Ajax and jQuery

**EVALUATION COPY: Not to be used in class.**

### Topics Covered

- Ajax and JSON Basics.
- The Problems and Solutions with Ajax.
- Getting data with Ajax.
- Sending data with Ajax.

### Introduction

Asynchronous JavaScript and XML (Ajax) was jQuery's killer app when jQuery was young. To simplify the process of using JavaScript to communicate with a server, jQuery introduced a number of simple methods that just worked, no matter what browser they ran in. Today, modern web browsers all feature support for the vanilla JavaScript Fetch API, which accomplishes the same thing as jQuery's Ajax methods, but without requiring a library to use.

In this lesson, we'll show you how to convert code that uses the Fetch API to use jQuery's Ajax methods. In the process, you'll also learn how to go the other way, which is probably the task you are more likely face.

**EVALUATION COPY: Not to be used in class.**



### 6.1. What is Ajax?

Ajax is the term that became popular to describe the use of HTTP and JavaScript to asynchronously get and send data and then use that data to update the browser DOM without reloading the web page. When it was new, Ajax usually involved XML data. Today, however, Ajax most often uses JSON data.

### ❖ 6.1.1. Working with JSON

JSON is a simple data format that resembles the object literal notation in JavaScript. For example, here's an object created using object literal notation:

```
const ajaxVocab = [  
  item: {  
    term: "Ajax",  
    definition: "Asynchronous JavaScript and XML"  
  },  
  item: {  
    term: "HTTP",  
    definition: "Hypertext Transfer Protocol"  
  },  
  item: {  
    term: "GET",  
    definition: "Requests a resource from the server."  
  },  
  item: {  
    term: "POST",  
    definition: "Requests that the server accept the enclosed data."  
  }  
];
```

Evaluation  
Copy

Here's a JSON representation of this data:

```
{[  
    "item": {  
        "term": "Ajax",  
        "definition": "Asynchronous JavaScript and XML"  
    },  
    "item": {  
        "term": "HTTP",  
        "definition": "Hypertext Transfer Protocol"  
    },  
    "item": {  
        "term": "GET",  
        "definition": "Requests a resource from the server."  
    },  
    "item": {  
        "term": "POST",  
        "definition": "Requests that the server accept the enclosed data."  
    }  
}]}
```

Notice the important differences between the two:

1. JavaScript objects need to be assigned to a variable, whereas JSON is a format for transmitting data, and is not assigned to a variable.
2. The left side of the colon in JSON data (the key) needs to be enclosed in quotes. This requirement ensures that the use of a reserved word in JSON data won't cause errors in any programs that consume the data.

The process of converting a JavaScript object into JSON data is called “stringifying.” The native JavaScript method for converting a JavaScript object to JSON data is `JSON.stringify()`.

The process of converting JSON data into a JavaScript object is called “parsing.” The native JavaScript method for making this conversion is `JSON.parse()`.

### ❖ 6.1.2. Ajax and Cross-Domain Issues

By default, browsers will only allow JavaScript code running in a web page to make HTTP requests to the same server that the web page was downloaded from. This is called the "same-origin policy" and it is a security restriction intended to isolate potentially malicious documents.

However, same-origin policy is a frequent cause of problems with web applications, and so it's important to know how to properly request data from a different server from the one where your HTML is hosted.

The best approach for allowing cross-origin requests is to use CORS, which stands for Cross-Origin Resource Sharing. CORS must be configured on the server you're sending requests to. CORS is an HTTP header that describes which origins are permitted to read particular data.

**EVALUATION COPY: Not to be used in class.**



## 6.2. jQuery's Ajax Methods

The generic `ajax()` method in jQuery can be used to make any type of HTTP request that you need. It has a number of parameters that make it extremely flexible and powerful. However, all of its options also make the `ajax()` method complex.

In its simplest form, `ajax()` can be used with no parameters:

```
$ajax()
```

This simplest form simply loads the current page and doesn't do anything with the result. To load data from a different URL, use the `url` parameter. To use the result, you can use one or more of the handler methods for ajax and pass in a callback function. The Ajax handler methods are `done()`, `fail()`, and `always()`.

Here's a simple example that retrieves data from a URL and then specifies handlers for that data:

```
$ajax({
  url: 'https://example.com/get-orders'
})
.done(function() {
  console.log("got it!");
})
.fail(function() {
  console.log("ajax error");
})
.always(function() {
  console.log("complete");
});
```

# Exercise 17: Form Validation with Ajax

 20 to 30 minutes

In this lesson, you'll write a script to submit coupon codes from an HTML form to a server using Ajax. The server will validate the codes and return a message to be displayed in the browser.

## Node and npm

This exercise requires that you have Node and npm installed. For instructions on installing Node and npm, see <https://www.webucator.com/article/nodejs-and-node-package-manager-npm/>.

1. Right-click the `Exercises/ajax` folder in Visual Studio Code's Explorer and select **Open in Integrated Terminal**.
2. Run `npm install` in the Terminal to install all of the dependencies of the Node server.
3. Run `node app` to launch the application, and then go to `http://localhost:8080` in a web browser.
4. Enter a value into the green box with the coupon input box, and then click the button to validate the coupon. Unless you're very lucky or looked at the code for the server already, you probably got a message that the coupon was invalid.
5. Enter the coupon code "zippy" into the coupon input field and validate it. You should get a message that the coupon is valid.



Coupon: Coupon validated

6. Open `Exercises/ajax/public/ice-cream.html` in your editor and add a `script` element to the header to include jQuery.
7. Open `Exercises/ajax/public/ajax.js` in your editor:

## Exercise Code 17.1: Exercises/ajax/public/ajax.js

---

```
1.  async function postData(data) {  
2.    const response = await fetch('/coupon-check', {  
3.      method: 'POST',  
4.      headers: {  
5.        'Content-Type': 'application/json'  
6.      },  
7.      body: JSON.stringify(data)  
8.    });  
9.    return response.text();  
10.  }  
-----Lines 11 through 28 Omitted-----
```

---

Right now, this file uses vanilla JavaScript. You're going to convert the function that does the Ajax in this file to jQuery.

8. Replace the `fetch()` method (including the `await` keyword) with the jQuery `$.ajax()` method:

```
const response = $.ajax('/coupon-check', ...
```

9. The second parameter of the `$.ajax()` method, like the second parameter of the `fetch()` method, is an options object. However, in jQuery, a couple of the property names are different.
10. Change the name of the `body` property in the options object to `data`.
11. Chain a `done(response)` method to the end of the `$.ajax()` method, and move the `return` statement inside of it, but return `response.text()` instead of `response.text()`.
12. Refresh `http://localhost:8080` in your browser. It should continue to work.



## Solution: Solutions/ajax/public/ajax.js

---

```
1.  async function postData(data) {  
2.      const response = $.ajax('/coupon-check', {  
3.          method: 'POST',  
4.          headers: {  
5.              'Content-Type': 'application/json'  
6.          },  
7.          data: JSON.stringify(data)  
8.      })  
9.      .done(function(response) {  
10.          return response;  
11.      });  
12.      return response;  
13.  }  
-----Lines 14 through 31 Omitted-----
```

Evaluation  
Copy

## Conclusion

In this lesson, you have learned how to use jQuery's Ajax methods, and how to convert code written using the Fetch API into jQuery.

# LESSON 7

## Converting from jQuery to JavaScript

**EVALUATION COPY: Not to be used in class.**

### Topics Covered

- Why convert?
- Practice, practice, practice!

### Introduction

In this lesson, you'll work step by step through the jQuery code for Mathificent and convert each function into vanilla JavaScript.

**EVALUATION COPY: Not to be used in class.**

\*

### 7.1. Why Convert from jQuery to JavaScript?

With modern JavaScript syntax and the changes that have been made to the language in recent years, it has become easier to replace jQuery code with vanilla JavaScript. Many of the issues that formerly made jQuery so beneficial, such as being able to write cross-browser compatible code, are no longer concerns as a result of improvements to browser support for JavaScript and the changes that have been made to the JavaScript language since JavaScript 5.1 and ES2015.

By rewriting jQuery code as vanilla JavaScript, you eliminate the download and runtime overhead associated with jQuery while also making your code compatible with more modern JavaScript libraries such as React, Vue, and Angular.

# Exercise 18: Getting Ready

 45 to 90 minutes

---

In this exercise, you will review the HTML and JavaScript files used to create Mathificent with jQuery.

## `index.html`

Open `Exercises/jquery-to-javascript/index.html` in your editor and spend some time reviewing the code. You won't need to make any changes to this file until the end of the lesson, at which point, you will be able to remove the jQuery script element.

## `mathificent.js`

Open `Exercises/jquery-to-javascript/scripts/mathificent.js` in your editor and spend some time reviewing the code. Pay particular attention to the comments.

Both files are shown on the pages that follow. You can review them here or open them in your editor and review them there. Go slowly through the code line by line. The first step to any code revision is getting a good understanding of how everything fits together.

## Exercise Code 18.1: Exercises/jquery-to-javascript/index.html

---

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.  <meta charset="UTF-8">
5.  <meta name="viewport" content="width=device-width, initial-scale=1">
6.  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">
7.  <link rel="stylesheet" href="styles/mathificent.css">
8.  <title>Mathificent</title>
9.  </head>
10. <body>
11. <header>
12.   <nav class="navbar navbar-expand-lg navbar-dark">
13.     <ul class="navbar-nav mr-auto text-left">
14.       <li class="nav-item active"><a href="/" class="nav-link">Home</a></li>
15.     </ul>
16.     <a href="/" class="navbar-brand">Mathificent</a>
17.   </nav>
18. </header>
19. <main>
20.   <section id="config-container">
21.     <h2>Mathificent</h2>
22.     <div class="row mx-1 my-3" value="x">
23.       <label for="operation" class="col fw-bold">Operation</label>
24.       <select id="operation" class="col form-control">
25.         </select>
26.       </div>
27.     <div class="row mx-1 my-3" value="10">
28.       <label for="max-number" class="col fw-bold">Maximum Number</label>
29.       <select id="max-number" class="col form-control">
30.         </select>
31.       </div>
32.     <div class="row mx-1 my-3">
33.       <button id="btn-play-button" class="form-control btn btn-primary">Play!</button>
34.     </div>
35.   </div>
36. </section>
37. <section id="game-container" class="text-center">
38.   <div id="scoreboard" class="row border-bottom">
39.     <div class="col px-3 text-left">
40.       <strong>Score: <output id="score">0</output></strong>
41.     </div>
42.     <div class="col px-3 text-right">
```

```
43.          <strong>Time Left: <output id="time-left"></output></strong>
44.      </div>
45.  </div>
46.  <div id="equation-container" class="text-secondary my-2 row">
47.      <div class="col-5"><output id="equation"></output></div>
48.      <div class="col-2">=</div>
49.      <div class="col-5"><output id="user-solution"></output></div>
50.  </div>
51.  <div id="game-buttons">
52.      <button class="btn btn-primary number-button" data-value="1">1</button>
53.      <button class="btn btn-primary number-button" data-value="2">2</button>
54.      <button class="btn btn-primary number-button" data-value="3">3</button>
55.      <button class="btn btn-primary number-button" data-value="4">4</button>
56.      <button class="btn btn-primary number-button" data-value="5">5</button>
57.      <button class="btn btn-primary number-button" data-value="6">6</button>
58.      <button class="btn btn-primary number-button" data-value="7">7</button>
59.      <button class="btn btn-primary number-button" data-value="8">8</button>
60.      <button class="btn btn-primary number-button" data-value="9">9</button>
61.      <button class="btn btn-primary number-button" data-value="0">0</button>
62.      <button id="btn-clear-button" class="btn btn-primary">Clear</button>
63.  </div>
64. </section>
65. <section id="times-up-container" class="text-center">
66.     <h2>Time's Up!</h2>
67.     <strong class="big">You Answered</strong>
68.     <div class="huge"><output id="final-score"></output></div>
69.     <strong class="big">Questions Correctly</strong>
70.     <button id="btn-play-again" class="btn btn-primary form-control m-1">
71.         Play Again with Same Settings
72.     </button>
73.     <button id="btn-change-settings" class="btn btn-secondary form-control m-1">
74.         Change Settings
75.     </button>
76. </section>
77. </main>
78. <footer class="navbar fixed-bottom">
79.     <a href="https://www.webucator.com" class="text-light">© 2022 Webucator</a>
80. </footer>
81. <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous">
82.     integrity="sha256-/xUj+3OJU5yExlq6GSYGSKh7tPXikynS7ogEvDej/m4="></script>
83. <script src="scripts/mathificent.js"></script>
84. </body>
85. </html>
```

## Exercise Code 18.2: Exercises/jquery-to-javascript/scripts/mathificent.js

---

```
1.  ****
2.  Global variables
3.  ****
4.  let high; // The highest number used in equations.
5.  let low = 0; // The lowest number used in equations.
6.  let operator; // The operator currently being used.
7.
8.
9.  ****
10. Utility Functions
11. ****
12. function randInt(low, high) {
13.   /* Utility function for getting random integer. */
14.   return Math.floor(Math.random() * (high - low + 1) + low);
15. }
16.
17.
18. ****
19. Common Functions
20. ****
21. function captureChangeViewEvents() {
22.   // Cache reused elements
23.   const configContainer = $('#config-container');
24.   const gameContainer = $('#game-container');
25.   const timesUpContainer = $('#times-up-container');
26.
27.   // Capture play button clicks
28.   $('#btn-play-button').click(function() {
29.     // Set global variable: high
30.     high = Number($('#max-number').val());
31.
32.     // Set global variable: operator
33.     switch ($('#operation').val()) {
34.       case 'addition':
35.         operator = '+';
36.         break;
37.       case 'subtraction':
38.         operator = '-';
39.         break;
40.       case 'multiplication':
41.         operator = 'x';
42.         break;
43.       case 'division':
44.         operator = '/';
```

```

45.      }
46.      switchSection(configContainer, gameContainer);
47.      play();
48.  });
49.
50.  // Capture play-again button clicks
51.  $('#btn-play-again').click(function() {
52.    switchSection(timesUpContainer, gameContainer);
53.    play();
54.  });
55.
56.  // Capture change-settings button clicks
57.  $('#btn-change-settings').click(function() {
58.    switchSection(timesUpContainer, configContainer);
59.  });
60. }
61.
62. function switchSection(hideSection, showSection) {
63.   // Hide one section and display another
64.   hideSection.toggle('display');
65.   showSection.toggle('display');
66.
67.   // If leaving game, turn off game event capturing
68.   if (hideSection.attr('id') === 'game-container') {
69.     captureGameEvents('off');
70.   }
71.
72.   // If starting game, turn on game event capturing
73.   if (showSection.attr('id') === 'game-container') {
74.     captureGameEvents('on');
75.   }
76. }
77.
78. ****
79. Config View Functions
80. ****
81. function populateSelects() {
82.   // Cache reused elements
83.   const operationSelect = $('#operation');
84.   const maxNumberSelect = $('#max-number');
85.
86.   // Populate operator options
87.   const operations = ['Addition', 'Subtraction', 'Multiplication', 'Division'];
88.
89.   $.each(operations, function(i, val) {

```

```

90.      $(`<option value="${val.toLowerCase()}">${val}</option>`)
91.          .appendTo(operationSelect);
92.      });
93.
94.      operationSelect.val('multiplication'); // Initial operator
95.      operationSelect.focus();
96.
97.      // Populate max number options
98.      for (let i=2; i <= 100; i++) {
99.          `<option value="${i}">${i}</option>`).appendTo(maxNumberSelect);
100.     }
101.     maxNumberSelect.val(10); // Initial max number
102. }
103.
104. ****
105. Game View Functions
106. ****
107. function answerChanged(value) {
108.     /*
109.         This is the callback function that gets called each time the user adds
110.         their answer. It gets the user's current answer, appends the new change,
111.         checks the result against the correct answer. If correct, it calls
112.         setEquation() to get a new equation.
113.     */
114.     // Cache reused elements
115.     const userSolutionElem = $('#user-solution');
116.     const scoreElem = $('#score');
117.
118.     const correctAnswer = getCorrectAnswer(operator);
119.     let score = Number(scoreElem.text());
120.     let userAnswer = Number(userSolutionElem.text());
121.
122.     if (userAnswer === 0) {
123.         // If userAnswer is 0, don't append, just change the value.
124.         // This prevents answers like "05"
125.         userSolutionElem.text(value);
126.     } else {
127.         // Append the new value to the user's previous answer.
128.         userSolutionElem.append(value);
129.     }
130.
131.     // Retrieve and convert the new answer to an integer.
132.     userAnswer = Number(userSolutionElem.text());
133.
134.     // Check if the answer is correct

```

```
135. if (correctAnswer === userAnswer) {
136.     score++;
137.     scoreElem.text(score.toString()); // Update scoreboard.
138.     userSolutionElem.text(''); // Clear user answer.
139.     setEquation(operator, low, high); // Get new equation.
140. };
141. }
142.
143. function captureGameEvents(onoff = 'on') {
144.     /*
145.         This is used to turn event capturing for game-play events on and off. Its
146.         primary purpose is for keyboard events, as the user can continue to press
147.         keys after the game view has disappeared.
148.     */
149.
150.     // Decide whether we are adding or removing events.
151.     // We no longer need this as we can pass onoff to the jQuery object
152.     // const eventAction = (onoff === 'on') ?
153.     //     'addEventListener' : 'removeEventListener';
154.
155.     // Add/Remove click event to all number buttons.
156.     $('section#game-container .number-button')[onoff]('click', numButtonClicked);
157.
158.     // Add/Remove click event to clear button.
159.     $('#btn-clear-button')[onoff]('click', clearButtonClicked);
160.
161.     // Add/Remove keyup events to enable keyboard play.
162.     $(document)[onoff]('keyup', keyPressed);
163. }
164.
165. function clearButtonClicked() {
166.     $('#user-solution').text('');
167. }
168.
169. function getCorrectAnswer(oper) {
170.     /* Calculate and return correct answer. */
171.     const num1 = Number($('#num1').text());
172.     const num2 = Number($('#num2').text());
173.     switch (oper) {
174.         case '+':
175.             return num1 + num2;
176.         case '-':
177.             return num1 - num2;
178.         case 'x':
179.             return num1 * num2;
```

```

180.     case '/':
181.         return num1 / num2;
182.     }
183. }
184.
185. function keyPressed(e) {
186.     /* Callback function to keyboard clicks. */
187.     // Cache reused elements
188.     const userSolutionElem = $('#user-solution');
189.
190.     const value = e.key;
191.     if (e.keyCode === 8 || e.keyCode == 46) { // backspace
192.         const currentAnswer = userSolutionElem.text();
193.         if (currentAnswer.length) {
194.             const newAnswer = currentAnswer.substring(0, currentAnswer.length - 1);
195.             userSolutionElem.text(newAnswer);
196.         }
197.     } else if (e.keyCode === 32) { // spacebar
198.         userSolutionElem.text('');
199.     } else if (e.keyCode >= 48 && e.keyCode <= 57) { // Number keys
200.         answerChanged(value);
201.     }
202. }
203.
204. function numButtonClicked(e) {
205.     /* Callback function to number button clicks. */
206.     const btn = $(e.target); // Which button?
207.     const value = btn.attr('data-value'); // Get the value of the clicked button.
208.     answerChanged(value); // Change the answer.
209. }
210.
211. function play() {
212.     /* Start playing. */
213.     $('#time-left').text(60); // seconds left
214.     $('#score').text(0);
215.     setEquation();
216.     startTimer();
217. }
218.
219. function setEquation() {
220.     /* Sets the equation. */
221.     // Cache reused elements
222.     const userSolutionElem = $('#user-solution');
223.     const equationElem = $('#equation');
224.

```

```

225. let num1 = randInt(low, high);
226. let num2 = randInt(low, high, true);
227. if (operator === '-') { // Subtract smaller number from bigger number.
228.     num1 = Math.max(num1, num2);
229.     num2 = Math.min(num1, num2);
230. } else if (operator === '/') {
231.     while (num2 === 0) { // No division by zero
232.         num2 = randInt(low, high);
233.     }
234.     // Make num1 the product of num1 x num2, so that the equation is num1/num2.
235.     // e.g., if the random numbers are 5 and 7, the equation will be 35 / 7.
236.     num1 = num1 * num2;
237. }
238.
239. // Create the spans to hold the operands
240. const spanNum1 = '<span id="num1">' + num1.toString() + '</span>';
241. const spanNum2 = '<span id="num2">' + num2.toString() + '</span>';
242.
243. // Clear the user answer.
244. userSolutionElem.text('');
245.
246. equationElem.html(spanNum1 + ' ' + operator + ' ' + spanNum2);
247. }
248.
249. function startTimer() {
250.     // Start the Timer Countdown.
251.     const timer = setInterval(function() {
252.         const timeLeft = $('#time-left');
253.
254.         let secondsLeft = Number(timeLeft.text());
255.         secondsLeft--;
256.         timeLeft.text(secondsLeft.toString());
257.
258.         if (secondsLeft < 10) { // Start changing background color.
259.             const a = (10 - secondsLeft) * .1;
260.             const bgColor = 'rgb(255,204,102,' + a + ')';
261.             $('body').css({
262.                 backgroundColor: bgColor
263.             });
264.         }
265.
266.         if (secondsLeft === 0) { // Times up.
267.             const score = Number($('#score').text());
268.             clearInterval(timer);
269.             timesUp(score);

```

```
270.      }
271.    }, 1000);
272.  }
273.
274. function timesUp(score) {
275.   $('body').css({
276.     backgroundColor: '#fff' // Reset background to white.
277.   });
278.
279.   // Show the times-up view.
280.   switchSection(
281.     $('#game-container'),
282.     $('#times-up-container')
283.   );
284.
285.   // Show the score.
286.   $('#final-score').text(score);
287. }
288.
289. ****
290. Initial Load
291. ****
292. $(function() {
293.   populateSelects();
294.   captureChangeViewEvents();
295. });
```

---

Evaluation  
Copy

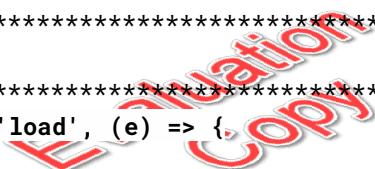
Now that you have thoroughly reviewed the code, update the **Initial Load** function at the end of `mathificent.js` to use the vanilla JavaScript method of waiting for the document to be ready.

## Solution: Solutions/jquery-to-javascript/scripts/mathificent-1.js

---

-----Lines 1 through 288 Omitted-----

```
289. /*****  
290. Initial Load  
291. ****/  
292. window.addEventListener('load', (e) => {  
293.   populateSelects();  
294.   captureChangeViewEvents();  
295. });
```



# Exercise 19: Converting the Common Functions

 25 to 40 minutes

Convert everything you can in the “Common Functions” section of `mathificent.js` from jQuery to vanilla JavaScript. This involves making changes to the following functions:

1. `captureChangeViewEvents()`
2. `switchSection()`

Keep `Exercises/jquery-to-javascript/index.html` open in your browser and test it as you make changes. It should continue to work after each change. Only one thing will break: when a game is finished, it will not switch to the times-up screen. You will fix that shortly.

## Solution: Solutions/jquery-to-javascript/scripts/mathificent-2.js

---

```
-----Lines 1 through 17 Omitted-----
18. /*****
19.     Common Functions
20. *****/
21. function captureChangeViewEvents() {
22.     // Cache reused elements
23.     const configContainer = document.getElementById('config-container');
24.     const gameContainer = document.getElementById('game-container');
25.     const timesUpContainer = document.getElementById('times-up-container');
26.
27.     // Capture play button clicks
28.     const playButton = document.getElementById('btn-play-button');
29.     playButton.addEventListener('click', function() {
30.         // Set global variable: high
31.         high = Number(document.getElementById('max-number').value);
32.
33.         // Set global variable: operator
34.         switch (document.getElementById('operation').value) {
35.             case 'addition':
36.                 operator = '+';
37.                 break;
38.             case 'subtraction':
39.                 operator = '-';
40.                 break;
41.             case 'multiplication':
42.                 operator = 'x';
43.                 break;
44.             case 'division':
45.                 operator = '/';
46.         }
47.         switchSection(configContainer, gameContainer);
48.         play();
49.     });
50.
51.     // Capture play-again button clicks
52.     const playAgainButton = document.getElementById('btn-play-again');
53.     playAgainButton.addEventListener('click', function() {
54.         switchSection(timesUpContainer, gameContainer);
55.         play();
56.     });
57.
58.     // Capture change-settings button clicks
59.     const changeSettingsButton = document.getElementById('btn-change-settings');
60.     changeSettingsButton.addEventListener('click', function() {
```

Evaluation  
Copy

```
61.     switchSection(timesUpContainer, configContainer);
62.   });
63. }
64.
65. function switchSection(hideSection, showSection) {
66.   // Hide one section and display another
67.   hideSection.style.display = 'none';
68.   showSection.style.display = 'block';
69.
70.   // If leaving game, turn off game event capturing
71.   if (hideSection.id === 'game-container') {
72.     captureGameEvents('off');
73.   }
74.
75.   // If starting game, turn on game event capturing
76.   if (showSection.id === 'game-container') {
77.     captureGameEvents('on');
78.   }
79. }
```

-----Lines 80 through 298 Omitted-----

---

## Code Explanation

---

One thing you will lose is the transition effect when moving from one view to another. jQuery effects are complex and require a significant amount of code to reproduce with vanilla JavaScript.

---

# Exercise 20: Converting the Config View Functions

 10 to 15 minutes

There is only one Config View function: `populateSelects()`. Convert all jQuery code you can find in that function to use vanilla JavaScript. This includes replacing the `$.each()` method with JavaScript arrays' `forEach()`.

Again, keep `Exercises/jquery-to-javascript/index.html` open in your browser and test it as you make changes. It should continue to work after each change.



## Solution: Solutions/jquery-to-javascript/scripts/mathificent-3.js

---

```
-----Lines 1 through 80 Omitted-----
81. /*****
82.   Config View Functions
83. *****/
84. function populateSelects() {
85.   // Cache reused elements
86.   const operationSelect = document.getElementById('operation');
87.   const maxNumberSelect = document.getElementById('max-number');
88.
89.   // Populate operator options
90.   const operations = ['Addition', 'Subtraction', 'Multiplication', 'Division'];
91.
92.   operations.forEach((operation) => {
93.     const option = document.createElement('option');
94.     option.setAttribute('value', operation.toLowerCase());
95.     option.innerHTML = operation;
96.     operationSelect.appendChild(option);
97.   })
98.
99.   operationSelect.value = 'multiplication'; // Initial operator
100.  operationSelect.focus();
101.
102.  // Populate max number options
103.  for (let i = 2; i <= 100; i++) {
104.    const option = document.createElement('option');
105.    option.setAttribute('value', i);
106.    option.innerHTML = i;
107.    maxNumberSelect.appendChild(option);
108.  }
109.
110.  maxNumberSelect.value = 10; // Initial Max Number
111. }

-----Lines 112 through 304 Omitted-----
```

# Exercise 21: Convert the Game View Functions

 60 to 90 minutes

Convert everything you can in the “Game View Functions” section of `mathificent.js` from jQuery to vanilla JavaScript. This involves making changes to the following functions:

1. `answerChanged()`
2. `captureGameEvents()`
3. `clearButtonClicked()`
4. `getCorrectAnswer()`
5. `keyPressed()`
6. `numButtonClicked()`
7. `play()`
8. `setEquation()`
9. `startTimer()`
10. `timesUp()`

Evaluation  
Copy

Again, keep `Exercises/jquery-to-javascript/index.html` open in your browser and test it as you make changes. It should continue to work after each change. While testing, you may find it useful to change the value of the `time-left` text to a smaller number (e.g., 15), so games go by more quickly.

When you’re done, there should be no jQuery left in the file, and everything should work as it did before you started converting to jQuery.

The solutions are shown on the following pages function by function and the final solution is in `Solutions/jquery-to-javascript/scripts/mathificent.js`.

## Solution: answerChanged()

---

```
-----Lines 1 through 115 Omitted-----  
116. function answerChanged(value) {  
117.     /*  
118.         This is the callback function that gets called each time the user adds  
119.         their answer. It gets the user's current answer, appends the new change,  
120.         checks the result against the correct answer. If correct, it calls  
121.         setEquation() to get a new equation.  
122.     */  
123.     // Cache reused elements  
124.     const userSolutionElem = document.getElementById('user-solution');  
125.     const scoreElem = document.getElementById('score');  
126.  
127.     const correctAnswer = getCorrectAnswer(operator);  
128.     let score = Number(scoreElem.innerText);  
129.     let userAnswer = Number(userSolutionElem.innerText);  
130.  
131.     if (userAnswer === 0) {  
132.         // If userAnswer is 0, don't append, just change the value.  
133.         // This prevents answers like "05"  
134.         userSolutionElem.innerText = value;  
135.     } else {  
136.         // Append the new value to the user's previous answer.  
137.         userSolutionElem.innerText += value;  
138.     }  
139.  
140.     // Retrieve and convert the new answer to an integer.  
141.     userAnswer = Number(userSolutionElem.innerText);  
142.  
143.     // Check if the answer is correct  
144.     if (correctAnswer === userAnswer) {  
145.         score++;  
146.         scoreElem.innerText = score.toString(); // Update scoreboard.  
147.         userSolutionElem.innerText = ''; // Clear user answer.  
148.         setEquation(operator, low, high); // Get new equation.  
149.     };  
150. }  
-----Lines 151 through 303 Omitted-----
```

## Solution: captureGameEvents()

---

```
-----Lines 1 through 151 Omitted-----  
152. function captureGameEvents(onoff = 'on') {  
153.   /*  
154.     This is used to turn event capturing for game-play events on and off. Its  
155.     primary purpose is for keyboard events, as the user can continue to press  
156.     keys after the game view has disappeared.  
157.   */  
158.   const numPadButtons = document.getElementsByClassName('number-button');  
159.  
160.   // Decide whether we are adding or removing events.  
161.   const eventAction = (onoff === 'on') ?  
162.     'addEventListener' : 'removeEventListener';  
163.  
164.   // Add/Remove click event to all number buttons  
165.   for (numBtn of numPadButtons) {  
166.     numBtn[eventAction]('click', numButtonClicked);  
167.   }  
168.  
169.   // Add/Remove click event to clear button  
170.   const clearButton = document.getElementById('btn-clear-button');  
171.   clearButton[eventAction]('click', clearButtonClicked);  
172.  
173.   // Add/Remove keyup events to enable keyboard play  
174.   document[eventAction]('keyup', keyPressed);  
175. }  
-----Lines 176 through 303 Omitted-----
```

---

## Solution: clearButtonClicked()

---

```
-----Lines 1 through 176 Omitted-----  
177. function clearButtonClicked() {  
178.   document.getElementById('user-solution').innerText = '';  
179. }  
-----Lines 180 through 303 Omitted-----
```

---

## Solution: getCorrectAnswer()

---

```
-----Lines 1 through 180 Omitted-----  
181. function getCorrectAnswer(oper) {  
182.     /* Calculate and return correct answer. */  
183.     const num1 = Number(document.getElementById('num1').innerText);  
184.     const num2 = Number(document.getElementById('num2').innerText);  
185.     switch (oper) {  
186.         case '+':  
187.             return num1 + num2;  
188.         case '-':  
189.             return num1 - num2;  
190.         case 'x':  
191.             return num1 * num2;  
192.         case '/':  
193.             return num1 / num2;  
194.     }  
195. }  
-----Lines 196 through 303 Omitted-----
```

---

## Solution: keyPressed()

---

*Evaluation  
Copy*

```
-----Lines 1 through 196 Omitted-----  
197. function keyPressed(e) {  
198.     /* Callback function to keyboard clicks. */  
199.     // Cache reused elements  
200.     const userSolutionElem = document.getElementById('user-solution');  
201.  
202.     const value = e.key;  
203.     if (e.keyCode === 8 || e.keyCode === 46) { // backspace  
204.         const currentAnswer = userSolutionElem.innerText;  
205.         if (currentAnswer.length) {  
206.             const newAnswer = currentAnswer.substring(0, currentAnswer.length - 1);  
207.             userSolutionElem.innerText = newAnswer;  
208.         }  
209.     } else if (e.keyCode === 32) { // spacebar  
210.         userSolutionElem.innerText = '';  
211.     } else if (e.keyCode >= 48 && e.keyCode <= 57) { // Number keys  
212.         answerChanged(value);  
213.     }  
214. }  
-----Lines 215 through 303 Omitted-----
```

---

## Solution: numButtonClicked()

---

```
-----Lines 1 through 215 Omitted-----  
216. function numButtonClicked(e) {  
217.   /* Callback function to number button clicks. */  
218.   const btn = e.target; // Which button?  
219.   const value = btn.dataset.value; // Get the value of the clicked button.  
220.   answerChanged(value); // Change the answer.  
221. }  
-----Lines 222 through 303 Omitted-----
```

---

## Solution: play()

---



```
-----Lines 1 through 222 Omitted-----  
223. function play() {  
224.   /* Start playing. */  
225.   document.getElementById('time-left').innerText = 60; // seconds left  
226.   document.getElementById('score').innerText = 0;  
227.   setEquation();  
228.   startTimer();  
229. }  
-----Lines 230 through 303 Omitted-----
```

---

## Solution: setEquation()

---

```
-----Lines 1 through 230 Omitted-----  
231. function setEquation() {  
232.     /* Sets the equation. */  
233.     // Cache reused elements  
234.     const userSolutionElem = document.getElementById('user-solution');  
235.     const equationElem = document.getElementById('equation');  
236.  
237.     let num1 = randInt(low, high);  
238.     let num2 = randInt(low, high, true);  
239.     if (operator === '-') { // Subtract smaller number from bigger number.  
240.         num1 = Math.max(num1, num2);  
241.         num2 = Math.min(num1, num2);  
242.     } else if (operator === '/') {  
243.         while (num2 === 0) { // No division by zero  
244.             num2 = randInt(low, high);  
245.         }  
246.         // Make num1 the product of num1 x num2, so that the equation is num1/num2.  
247.         // e.g., if the random numbers are 5 and 7, the equation will be 35 / 7.  
248.         num1 = num1 * num2;  
249.     }  
250.  
251.     // Create the spans to hold the operands  
252.     const spanNum1 = '<span id="num1">' + num1.toString() + '</span>';  
253.     const spanNum2 = '<span id="num2">' + num2.toString() + '</span>';  
254.  
255.     // Clear the user answer.  
256.     userSolutionElem.innerText = '';  
257.  
258.     equationElem.innerHTML = spanNum1 + ' ' + operator + ' ' + spanNum2;  
259. }
```

-----Lines 260 through 303 Omitted-----

---

## Solution: startTimer()

---

```
-----Lines 1 through 260 Omitted-----  
261. function startTimer() {  
262.     // Start the Timer Countdown.  
263.     const timer = setInterval(function() {  
264.         const timeLeft = document.getElementById('time-left');  
265.  
266.         let secondsLeft = Number(timeLeft.innerText);  
267.         secondsLeft--;  
268.         timeLeft.innerText = secondsLeft.toString();  
269.  
270.         if (secondsLeft < 10) { // Start changing background color.  
271.             const a = (10 - secondsLeft) * .1;  
272.             const bgColor = 'rgb(255,204,102,' + a + ')';  
273.             document.body.style.backgroundColor = bgColor;  
274.         }  
275.  
276.         if (secondsLeft === 0) { // Times up.  
277.             const score = Number(document.getElementById('score').innerText);  
278.             clearInterval(timer);  
279.             timesUp(score);  
280.         }  
281.     }, 1000);  
282. }  
-----Lines 283 through 303 Omitted-----
```

Evaluation  
Copy

---

## Solution: timesUp()

---

```
-----Lines 1 through 283 Omitted-----  
284. function timesUp(score) {  
285.     document.body.style.backgroundColor = '#fff'; // Reset background to white.  
286.  
287.     // Show the times-up view.  
288.     switchSection(  
289.         document.getElementById('game-container'),  
290.         document.getElementById('times-up-container'))  
291.     );  
292.  
293.     // Show the score.  
294.     document.getElementById('final-score').innerText = score;  
295. }  
-----Lines 296 through 303 Omitted-----
```

---

## Code Explanation

---

Congratulations! You are done. You can now remove the jQuery script element from the `index.html` file. After doing so, test out your page one more time. It should continue to work.

---

## Conclusion

In this lesson, you have converted the Mathificent game from jQuery to vanilla JavaScript.

Evaluation  
Copy

# LESSON 8

## Converting from JavaScript to jQuery

EVALUATION COPY: Not to be used in class.

### Topics Covered

- Why convert?
- Practice, practice, practice!

### Introduction

In this lesson, you will work step by step through the vanilla JavaScript code for Mathificent and convert each function into jQuery.

Evaluation  
Copy

EVALUATION COPY: Not to be used in class.

\*

### 8.1. Why Convert to jQuery?

Because jQuery contains so many pre-built functions, converting a program from vanilla JavaScript to jQuery can reduce the amount of custom code in a program. This can reduce the potential for bugs, and it can result in more legible code. Of course, this is less true than it used to be. Today, if you're writing new code in jQuery, it's probably because you are working on a site that was built using jQuery. If you're doing a conversion, it's likely *from* jQuery *to* JavaScript (see page 95) or to some more modern JavaScript framework like React, Vue, or Angular. So, why convert from JavaScript to jQuery? Mostly just to help you learn jQuery well enough to maintain and update legacy jQuery code.

# Exercise 22: Getting Ready

 45 to 90 minutes

---

In this exercise, you will review the HTML and JavaScript files used to create Mathificent.

## **index.html**

Open `Exercises/javascript-to-jquery/index.html` in your editor and spend some time reviewing the code. You will only need to make one change to this file: Add the minified version of the latest jQuery script element, which you can get at <https://code.jquery.com/>. Paste it immediately before the `mathificent.js` script element.

## **mathificent.js**

Open `Exercises/javascript-to-jquery/scripts/mathificent.js` in your editor and spend some time reviewing the code. Pay particular attention to the comments. After you have a good understanding of how the code works, make one simple change: Update the `window.addEventListener('load', ...)` method at the end of `mathificent.js` to the jQuery method of waiting for the document to be ready.

Both files are shown on the pages that follow. You can review them here or open them in your editor and review them there. Go slowly through the code line by line. The first step to any code revision is getting a good understanding of how everything fits together.

## Exercise Code 22.1: Exercises/javascript-to-jquery/index.html

---

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.  <head>
4.  <meta charset="UTF-8">
5.  <meta name="viewport" content="width=device-width, initial-scale=1">
6.  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">
7.  <link rel="stylesheet" href="styles/mathificent.css">
8.  <title>Mathificent</title>
9.  </head>
10. <body>
11. <header>
12.   <nav class="navbar navbar-expand-lg navbar-dark">
13.     <ul class="navbar-nav mr-auto text-left">
14.       <li class="nav-item active"><a href="/" class="nav-link">Home</a></li>
15.     </ul>
16.     <a href="/" class="navbar-brand">Mathificent</a>
17.   </nav>
18. </header>
19. <main>
20.   <section id="config-container">
21.     <h2>Mathificent</h2>
22.     <div class="row mx-1 my-3" value="x">
23.       <label for="operation" class="col fw-bold">Operation</label>
24.       <select id="operation" class="col form-control">
25.         </select>
26.       </div>
27.     <div class="row mx-1 my-3" value="10">
28.       <label for="max-number" class="col fw-bold">Maximum Number</label>
29.       <select id="max-number" class="col form-control">
30.         </select>
31.       </div>
32.     <div class="row mx-1 my-3">
33.       <button id="btn-play-button" class="form-control btn btn-primary">Play!</button>
34.     </div>
35.   </div>
36. </section>
37. <section id="game-container" class="text-center">
38.   <div id="scoreboard" class="row border-bottom">
39.     <div class="col px-3 text-left">
40.       <strong>Score: <output id="score">0</output></strong>
41.     </div>
42.     <div class="col px-3 text-right">
```

```
43.          <strong>Time Left: <output id="time-left"></output></strong>
44.      </div>
45.  </div>
46.  <div id="equation-container" class="text-secondary my-2 row">
47.      <div class="col-5"><output id="equation"></output></div>
48.      <div class="col-2">=</div>
49.      <div class="col-5"><output id="user-solution"></output></div>
50.  </div>
51.  <div id="game-buttons">
52.      <button class="btn btn-primary number-button" data-value="1">1</button>
53.      <button class="btn btn-primary number-button" data-value="2">2</button>
54.      <button class="btn btn-primary number-button" data-value="3">3</button>
55.      <button class="btn btn-primary number-button" data-value="4">4</button>
56.      <button class="btn btn-primary number-button" data-value="5">5</button>
57.      <button class="btn btn-primary number-button" data-value="6">6</button>
58.      <button class="btn btn-primary number-button" data-value="7">7</button>
59.      <button class="btn btn-primary number-button" data-value="8">8</button>
60.      <button class="btn btn-primary number-button" data-value="9">9</button>
61.      <button class="btn btn-primary number-button" data-value="0">0</button>
62.      <button id="btn-clear-button" class="btn btn-primary">Clear</button>
63.  </div>
64. </section>
65. <section id="times-up-container" class="text-center">
66.     <h2>Time's Up!</h2>
67.     <strong class="big">You Answered</strong>
68.     <div class="huge"><output id="final-score"></output></div>
69.     <strong class="big">Questions Correctly</strong>
70.     <button id="btn-play-again" class="btn btn-primary form-control m-1">
71.         Play Again with Same Settings
72.     </button>
73.     <button id="btn-change-settings" class="btn btn-secondary form-control m-1">
74.         Change Settings
75.     </button>
76. </section>
77. </main>
78. <footer class="navbar fixed-bottom">
79.     <a href="https://www.webucator.com" class="text-light">© 2022 Webucator</a>
80. </footer>
81. <script src="scripts/mathificent.js"></script>
82. </body>
83. </html>
```

---

## Exercise Code 22.2: Exercises/javascript-to-jquery/scripts/mathificent.js

---

```
1.  ****
2.  Global variables
3.  ****
4.  let high; // The highest number used in equations.
5.  let low = 0; // The lowest number used in equations.
6.  let operator; // The operator currently being used.
7.
8.
9.  ****
10. Utility Functions
11. ****
12. function randInt(low, high) {
13.   /* Utility function for getting random integer. */
14.   return Math.floor(Math.random() * (high - low + 1) + low);
15. }
16.
17.
18. ****
19. Common Functions
20. ****
21. function captureChangeViewEvents() {
22.   // Cache reused elements
23.   const configContainer = document.getElementById('config-container');
24.   const gameContainer = document.getElementById('game-container');
25.   const timesUpContainer = document.getElementById('times-up-container');
26.
27.   // Capture play button clicks
28.   const playButton = document.getElementById('btn-play-button');
29.   playButton.addEventListener('click', function() {
30.     // Set global variable: high
31.     high = Number(document.getElementById('max-number').value);
32.
33.     // Set global variable: operator
34.     switch (document.getElementById('operation').value) {
35.       case 'addition':
36.         operator = '+';
37.         break;
38.       case 'subtraction':
39.         operator = '-';
40.         break;
41.       case 'multiplication':
42.         operator = 'x';
43.         break;
44.       case 'division':
```

```

45.         operator = '/';
46.     }
47.     switchSection(configContainer, gameContainer);
48.     play();
49. });
50.
51. // Capture play-again button clicks
52. const playAgainButton = document.getElementById('btn-play-again');
53. playAgainButton.addEventListener('click', function() {
54.     switchSection(timesUpContainer, gameContainer);
55.     play();
56. });
57.
58. // Capture change-settings button clicks
59. const changeSettingsButton = document.getElementById('btn-change-settings');
60. changeSettingsButton.addEventListener('click', function() {
61.     switchSection(timesUpContainer, configContainer);
62. });
63. }
64.
65. function switchSection(hideSection, showSection) {
66.     // Hide one section and display another
67.     hideSection.style.display = 'none';
68.     showSection.style.display = 'block';
69.
70.     // If leaving game, turn off game event capturing
71.     if (hideSection.id === 'game-container') {
72.         captureGameEvents('off');
73.     }
74.
75.     // If starting game, turn on game event capturing
76.     if (showSection.id === 'game-container') {
77.         captureGameEvents('on');
78.     }
79. }
80.
81. *****
82.     Config View Functions
83. *****
84. function populateSelects() {
85.     // Cache reused elements
86.     const operationSelect = document.getElementById('operation');
87.     const maxNumberSelect = document.getElementById('max-number');
88.
89.     // Populate operator options

```

```

90.    const operations = ['Addition', 'Subtraction', 'Multiplication', 'Division'];
91.
92.    operations.forEach((operation) => {
93.        const option = document.createElement('option');
94.        option.setAttribute('value', operation.toLowerCase());
95.        option.innerHTML = operation;
96.        operationSelect.appendChild(option);
97.    })
98.
99.    operationSelect.value = 'multiplication'; // Initial operator
100.   operationSelect.focus();
101.
102. // Populate max number options
103. for (let i = 2; i <= 100; i++) {
104.     const option = document.createElement('option');
105.     option.setAttribute('value', i);
106.     option.innerHTML = i;
107.     maxNumberSelect.appendChild(option);
108. }
109.
110. maxNumberSelect.value = 10; // Initial Max Number
111. }
112.
113. ****
114. Game View Functions
115. ****
116. function answerChanged(value) {
117. /*
118.     This is the callback function that gets called each time the user adds
119.     their answer. It gets the user's current answer, appends the new change,
120.     checks the result against the correct answer. If correct, it calls
121.     setEquation() to get a new equation.
122. */
123. // Cache reused elements
124. const userSolutionElem = document.getElementById('user-solution');
125. const scoreElem = document.getElementById('score');
126.
127. const correctAnswer = getCorrectAnswer(operator);
128. let score = Number(scoreElem.innerText);
129. let userAnswer = Number(userSolutionElem.innerText);
130.
131. if (userAnswer === 0) {
132.     // If userAnswer is 0, don't append, just change the value.
133.     // This prevents answers like "05"
134.     userSolutionElem.innerText = value;

```

```
135. } else {
136.     // Append the new value to the user's previous answer.
137.     userSolutionElem.innerHTML += value;
138. }
139.
140. // Retrieve and convert the new answer to an integer.
141. userAnswer = Number(userSolutionElem.innerHTML);
142.
143. // Check if the answer is correct
144. if (correctAnswer === userAnswer) {
145.     score++;
146.     scoreElem.innerHTML = score.toString(); // Update scoreboard.
147.     userSolutionElem.innerHTML = ''; // Clear user answer.
148.     setEquation(operator, low, high); // Get new equation.
149. }
150. }
151.
152. function captureGameEvents(onoff = 'on') {
153. /*
154.     This is used to turn event capturing for game-play events on and off. Its
155.     primary purpose is for keyboard events, as the user can continue to press
156.     keys after the game view has disappeared.
157. */
158. const numPadButtons = document.getElementsByClassName('number-button');
159.
160. // Decide whether we are adding or removing events.
161. const eventAction = (onoff === 'on') ?
162.     'addEventListener' : 'removeEventListener';
163.
164. // Add/Remove click event to all number buttons
165. for (numBtn of numPadButtons) {
166.     numBtn[eventAction]('click', numButtonClicked);
167. }
168.
169. // Add/Remove click event to clear button
170. const clearButton = document.getElementById('btn-clear-button');
171. clearButton[eventAction]('click', clearButtonClicked);
172.
173. // Add/Remove keyup events to enable keyboard play
174. document[eventAction]('keyup', keyPressed);
175. }
176.
177. function clearButtonClicked() {
178.     document.getElementById('user-solution').innerHTML = '';
179. }
```

```

180.
181. function getCorrectAnswer(oper) {
182.   /* Calculate and return correct answer. */
183.   const num1 = Number(document.getElementById('num1').innerText);
184.   const num2 = Number(document.getElementById('num2').innerText);
185.   switch (oper) {
186.     case '+':
187.       return num1 + num2;
188.     case '-':
189.       return num1 - num2;
190.     case 'x':
191.       return num1 * num2;
192.     case '/':
193.       return num1 / num2;
194.   }
195. }
196.
197. function keyPressed(e) {
198.   /* Callback function to keyboard clicks. */
199.   // Cache reused elements
200.   const userSolutionElem = document.getElementById('user-solution');
201.
202.   const value = e.key;
203.   if (e.keyCode === 8 || e.keyCode === 46) { // backspace
204.     const currentAnswer = userSolutionElem.innerText;
205.     if (currentAnswer.length) {
206.       const newAnswer = currentAnswer.substring(0, currentAnswer.length - 1);
207.       userSolutionElem.innerText = newAnswer;
208.     }
209.   } else if (e.keyCode === 32) { // spacebar
210.     userSolutionElem.innerText = '';
211.   } else if (e.keyCode >= 48 && e.keyCode <= 57) { // Number keys
212.     answerChanged(value);
213.   }
214. }
215.
216. function numButtonClicked(e) {
217.   /* Callback function to number button clicks. */
218.   const btn = e.target; // Which button?
219.   const value = btn.dataset.value; // Get the value of the clicked button.
220.   answerChanged(value); // Change the answer.
221. }
222.
223. function play() {
224.   /* Start playing. */

```

```

225. document.getElementById('time-left').innerText = 60; // seconds left
226. document.getElementById('score').innerText = 0;
227. setEquation();
228. startTimer();
229. }
230.
231. function setEquation() {
232.   /* Sets the equation. */
233.   // Cache reused elements
234.   const userSolutionElem = document.getElementById('user-solution');
235.   const equationElem = document.getElementById('equation');
236.
237.   let num1 = randInt(low, high);
238.   let num2 = randInt(low, high, true);
239.   if (operator === '-') { // Subtract smaller number from bigger number.
240.     num1 = Math.max(num1, num2);
241.     num2 = Math.min(num1, num2);
242.   } else if (operator === '/') {
243.     while (num2 === 0) { // No division by zero
244.       num2 = randInt(low, high);
245.     }
246.     // Make num1 the product of num1 x num2, so that the equation is num1/num2.
247.     // e.g., if the random numbers are 5 and 7, the equation will be 35 / 7.
248.     num1 = num1 * num2;
249.   }
250.
251. // Create the spans to hold the operands
252. const spanNum1 = '<span id="num1">' + num1.toString() + '</span>';
253. const spanNum2 = '<span id="num2">' + num2.toString() + '</span>';
254.
255. // Clear the user answer.
256. userSolutionElem.innerText = '';
257.
258. equationElem.innerHTML = spanNum1 + ' ' + operator + ' ' + spanNum2;
259. }
260.
261. function startTimer() {
262.   // Start the Timer Countdown.
263.   const timer = setInterval(function() {
264.     const timeLeft = document.getElementById('time-left');
265.
266.     let secondsLeft = Number(timeLeft.innerText);
267.     secondsLeft--;
268.     timeLeft.innerText = secondsLeft.toString();
269.

```

```

270.     if (secondsLeft < 10) { // Start changing background color.
271.         const a = (10 - secondsLeft) * .1;
272.         const bgColor = `rgb(255,204,102, ${a})`;
273.         document.body.style.backgroundColor = bgColor;
274.     }
275.
276.     if (secondsLeft === 0) { // Times up.
277.         const score = Number(document.getElementById('score').innerText);
278.         clearInterval(timer);
279.         timesUp(score);
280.     }
281. }, 1000);
282. }
283.
284. function timesUp(score) {
285.     document.body.style.backgroundColor = '#fff'; // Reset background to white.
286.
287.     // Show the times-up view.
288.     switchSection(
289.         document.getElementById('game-container'),
290.         document.getElementById('times-up-container')
291.     );
292.
293.     // Show the score.
294.     document.getElementById('final-score').innerText = score;
295. }
296.
297. ****
298. Initial Load
299. ****
300. window.addEventListener('load', (e) => {
301.     populateSelects();
302.     captureChangeViewEvents();
303. });

```

---

Now that you have thoroughly reviewed the code, make the two small changes:

1. `index.html` – Add the minified version of the latest jQuery script element.
2. `mathificent.js` – Update the `window.addEventListener('load', ...)` method at the end of `mathificent.js` to the jQuery method of waiting for the document to be ready.

## Solution: Solutions/javascript-to-jquery/index.html

---

```
-----Lines 1 through 5 Omitted-----
6. <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">
7. <link rel="stylesheet" href="styles/mathificent.css">
8. <title>Mathificent</title>
9. </head>
10. <body>
-----Lines 12 through 80 Omitted-----
11. <script src="https://code.jquery.com/jquery-3.6.0.min.js" crossorigin="anonymous" integrity="sha256-/xUj+30JU5yExlq6GSYGSKh7tPXikynS7ogEvDejm4=></script>
12. <script src="scripts/mathificent.js"></script>
13. </body>
14. </html>
```

*Evaluation  
Copy*

---

## Solution: Solutions/javascript-to-jquery/scripts/mathificent-1.js

---

```
-----Lines 1 through 296 Omitted-----
297. ****
298. Initial Load
299. ****
300. $(function() {
301.   populateSelects();
302.   captureChangeViewEvents();
303. });
```

---

# Exercise 23: Converting the Common Functions

 25 to 40 minutes

Convert everything you can in the “Common Functions” section of `mathificent.js` to jQuery. This involves making changes to the following functions:

1. `captureChangeViewEvents()`
2. `switchSection()`

Keep `Exercises/javascript-to-jquery/index.html` open in your browser and test it as you make changes. It should continue to work after each change. Only one thing will break: when a game is finished, it will not switch to the times-up screen. You will fix that shortly.

## Solution: Solutions/javascript-to-jquery/scripts/mathificent-2.js

---

```
-----Lines 1 through 17 Omitted-----
18. /*****
19.     Common Functions
20. *****/
21. function captureChangeEvent() {
22.     // Cache reused elements
23.     const configContainer = $('#config-container');
24.     const gameContainer = $('#game-container');
25.     const timesUpContainer = $('#times-up-container');
26.
27.     // Capture play button clicks
28.     $('#btn-play-button').click(function() {
29.         // Set global variable: high
30.         high = Number($('#max-number').val());
31.
32.         // Set global variable: operator
33.         switch ($('#operation').val()) {
34.             case 'addition':
35.                 operator = '+';
36.                 break;
37.             case 'subtraction':
38.                 operator = '-';
39.                 break;
40.             case 'multiplication':
41.                 operator = 'x';
42.                 break;
43.             case 'division':
44.                 operator = '/';
45.         }
46.         switchSection(configContainer, gameContainer);
47.         play();
48.     });
49.
50.     // Capture play-again button clicks
51.     $('#btn-play-again').click(function() {
52.         switchSection(timesUpContainer, gameContainer);
53.         play();
54.     });
55.
56.     // Capture change-settings button clicks
57.     $('#btn-change-settings').click(function() {
58.         switchSection(timesUpContainer, configContainer);
59.     });
60. }
```

Evaluation  
Copy

```
61.  
62. function switchSection(hideSection, showSection) {  
63.     // Hide one section and display another  
64.     hideSection.toggle('display');  
65.     showSection.toggle('display');  
66.  
67.     // If leaving game, turn off game event capturing  
68.     if (hideSection.attr('id') === 'game-container') {  
69.         captureGameEvents('off');  
70.     }  
71.  
72.     // If starting game, turn on game event capturing  
73.     if (showSection.attr('id') === 'game-container') {  
74.         captureGameEvents('on');  
75.     }  
76. }
```

-----Lines 77 through 300 Omitted-----

---

## Code Explanation

---

One gain you get for free is the transition effect when moving from one view to another. jQuery effects are complex and require a significant amount of code to reproduce with vanilla JavaScript.

---

# Exercise 24: Converting the Config View Functions

 10 to 15 minutes

There is only one Config View function: `populateSelects()`. Convert everything you can in that function to use jQuery. This includes replacing the `forEach()` method with `$.each()`.

Again, keep `Exercises/javascript-to-jquery/index.html` open in your browser and test it as you make changes. It should continue to work after each change.



## Solution: Solutions/javascript-to-jquery/scripts/mathificent-3.js

---

```
-----Lines 1 through 77 Omitted-----  
78. /*****  
79.     Config View Functions  
80. *****  
81. function populateSelects() {  
82.     // Cache reused elements  
83.     const operationSelect = $('#operation');  
84.     const maxNumberSelect = $('#max-number');  
85.  
86.     // Populate operator options  
87.     const operations = ['Addition', 'Subtraction', 'Multiplication', 'Division'];  
88.  
89.     $.each(operations, function(i, val) {  
90.         `<option value="${val.toLowerCase()}">${val}</option>`)  
91.         .appendTo(operationSelect);  
92.     });  
93.  
94.     operationSelect.val('multiplication'); // Initial operator  
95.     operationSelect.focus();  
96.  
97.     // Populate max number options  
98.     for (let i=2; i <= 100; i++) {  
99.         `<option value="${i}">${i}</option>`).appendTo(maxNumberSelect);  
100.    }  
101.    maxNumberSelect.val(10); // Initial max number  
102. }  
-----Lines 103 through 294 Omitted-----
```

# Exercise 25: Convert the Game View Functions

 60 to 90 minutes

Convert everything you can in the “Game View Functions” section of `mathificent.js` to jQuery. This involves making changes to the following functions:

1. `answerChanged()`
2. `captureGameEvents()`
3. `clearButtonClicked()`
4. `getCorrectAnswer()`
5. `keyPressed()`
6. `numButtonClicked()`
7. `play()`
8. `setEquation()`
9. `startTimer()`
10. `timesUp()`

Evaluation  
Copy

Again, keep `Exercises/javascript-to-jquery/index.html` open in your browser and test it as you make changes. It should continue to work after each change. While testing, you may find it useful to change the value of the `time-left` text to a smaller number (e.g., 15), so games go by more quickly.

When you’re done, everything should work as it did before you started converting to jQuery.

The solutions are shown on the following pages function by function and the final solution is in `Solutions/javascript-to-jquery/scripts/mathificent.js`.

## Solution: answerChanged()

---

```
-----Lines 1 through 106 Omitted-----  
107. function answerChanged(value) {  
108.     /*  
109.         This is the callback function that gets called each time the user adds  
110.         their answer. It gets the user's current answer, appends the new change,  
111.         checks the result against the correct answer. If correct, it calls  
112.         setEquation() to get a new equation.  
113.     */  
114.     // Cache reused elements  
115.     const userSolutionElem = $('#user-solution');  
116.     const scoreElem = $('#score');  
117.  
118.     const correctAnswer = getCorrectAnswer(operator);  
119.     let score = Number(scoreElem.text());  
120.     let userAnswer = Number(userSolutionElem.text());  
121.  
122.     if (userAnswer === 0) {  
123.         // If userAnswer is 0, don't append, just change the value.  
124.         // This prevents answers like "05"  
125.         userSolutionElem.text(value);  
126.     } else {  
127.         // Append the new value to the user's previous answer.  
128.         userSolutionElem.append(value);  
129.     }  
130.  
131.     // Retrieve and convert the new answer to an integer.  
132.     userAnswer = Number(userSolutionElem.text());  
133.  
134.     // Check if the answer is correct  
135.     if (correctAnswer === userAnswer) {  
136.         score++;  
137.         scoreElem.text(score.toString()); // Update scoreboard.  
138.         userSolutionElem.text(''); // Clear user answer.  
139.         setEquation(operator, low, high); // Get new equation.  
140.     };  
141. }  
-----Lines 142 through 295 Omitted-----
```

## Solution: captureGameEvents()

---

```
-----Lines 1 through 142 Omitted-----  
143. function captureGameEvents(onoff = 'on') {  
144.     /*  
145.      This is used to turn event capturing for game-play events on and off. Its  
146.      primary purpose is for keyboard events, as the user can continue to press  
147.      keys after the game view has disappeared.  
148.     */  
149.  
150.    // Decide whether we are adding or removing events.  
151.    // We no longer need this as we can pass onoff to the jQuery object  
152.    // const eventAction = (onoff === 'on') ?  
153.    //   'addEventListener' : 'removeEventListener';  
154.  
155.    // Add/Remove click event to all number buttons.  
156.    $('section#game-container .number-button')[onoff]('click', numButtonClicked);  
157.  
158.    // Add/Remove click event to clear button.  
159.    $('#btn-clear-button')[onoff]('click', clearButtonClicked);  
160.  
161.    // Add/Remove keyup events to enable keyboard play.  
162.    $(document)[onoff]('keyup', keyPressed);  
163. }
```

-----Lines 164 through 295 Omitted-----

---

## Solution: clearButtonClicked()

---

```
-----Lines 1 through 164 Omitted-----  
165. function clearButtonClicked() {  
166.     $('#user-solution').text('');  
167. }  
-----Lines 168 through 295 Omitted-----
```

---

## Solution: getCorrectAnswer()

---

```
-----Lines 1 through 168 Omitted-----  
169. function getCorrectAnswer(oper) {  
170.   /* Calculate and return correct answer. */  
171.   const num1 = Number($('#num1').text());  
172.   const num2 = Number($('#num2').text());  
173.   switch (oper) {  
174.     case '+':  
175.       return num1 + num2;  
176.     case '-':  
177.       return num1 - num2;  
178.     case 'x':  
179.       return num1 * num2;  
180.     case '/':  
181.       return num1 / num2;  
182.   }  
183. }  
-----Lines 184 through 295 Omitted-----
```

---

## Solution: keyPressed()

---

Evaluation  
Copy

```
-----Lines 1 through 184 Omitted-----  
185. function keyPressed(e) {  
186.   /* Callback function to keyboard clicks. */  
187.   // Cache reused elements  
188.   const userSolutionElem = $('#user-solution');  
189.  
190.   const value = e.key;  
191.   if (e.keyCode === 8 || e.keyCode == 46) { // backspace  
192.     const currentAnswer = userSolutionElem.text();  
193.     if (currentAnswer.length) {  
194.       const newAnswer = currentAnswer.substring(0, currentAnswer.length - 1);  
195.       userSolutionElem.text(newAnswer);  
196.     }  
197.   } else if (e.keyCode === 32) { // spacebar  
198.     userSolutionElem.text('');  
199.   } else if (e.keyCode >= 48 && e.keyCode <= 57) { // Number keys  
200.     answerChanged(value);  
201.   }  
202. }  
-----Lines 203 through 295 Omitted-----
```

---

## Solution: numButtonClicked()

---

-----Lines 1 through 203 Omitted-----

```
204. function numButtonClicked(e) {  
205.   /* Callback function to number button clicks. */  
206.   const btn = $(e.target); // Which button?  
207.   const value = btn.attr('data-value'); // Get the value of the clicked button.  
208.   answerChanged(value); // Change the answer.  
209. }
```

-----Lines 210 through 295 Omitted-----

---

## Solution: play()

---

Evaluation  
Copy

-----Lines 1 through 210 Omitted-----

```
211. function play() {  
212.   /* Start playing. */  
213.   $('#time-left').text(60); // seconds left  
214.   $('#score').text(0);  
215.   setEquation();  
216.   startTimer();  
217. }
```

-----Lines 218 through 295 Omitted-----

---

## Solution: setEquation()

---

```
-----Lines 1 through 218 Omitted-----  
219. function setEquation() {  
220.   /* Sets the equation. */  
221.   // Cache reused elements  
222.   const userSolutionElem = $('#user-solution');  
223.   const equationElem = $('#equation');  
224.  
225.   let num1 = randInt(low, high);  
226.   let num2 = randInt(low, high, true);  
227.   if (operator === '-') { // Subtract smaller number from bigger number.  
228.     num1 = Math.max(num1, num2);  
229.     num2 = Math.min(num1, num2);  
230.   } else if (operator === '/') {  
231.     while (num2 === 0) { // No division by zero  
232.       num2 = randInt(low, high);  
233.     }  
234.     // Make num1 the product of num1 x num2, so that the equation is num1/num2.  
235.     // e.g., if the random numbers are 5 and 7, the equation will be 35 / 7.  
236.     num1 = num1 * num2;  
237.   }  
-----Lines 238 through 295 Omitted-----
```

## Solution: startTimer()

---

```
-----Lines 1 through 248 Omitted-----  
249. function startTimer() {  
250.     // Start the Timer Countdown.  
251.     const timer = setInterval(function() {  
252.         const timeLeft = $('#time-left');  
253.         let secondsLeft = Number(timeLeft.text());  
254.         secondsLeft--;  
255.         timeLeft.text(secondsLeft.toString());  
256.         if (secondsLeft < 10) { // Start changing background color.  
257.             const a = (10 - secondsLeft) * .1;  
258.             const bgColor = 'rgb(255,204,102,' + a + ')';  
259.             $('body').css({  
260.                 backgroundColor: bgColor  
261.             });  
262.         }  
263.         if (secondsLeft === 0) { // Times up.  
264.             const score = Number($('#score').text());  
265.             clearInterval(timer);  
266.             timesUp(score);  
267.         }  
268.     }, 1000);  
269. }  
270. -----Lines 273 through 295 Omitted-----  
271. }  
272. }
```

## Solution: timesUp()

---

```
-----Lines 1 through 273 Omitted-----  
274. function timesUp(score) {  
275.   $('body').css({  
276.     backgroundColor: '#fff' // Reset background to white.  
277.   });  
278.  
279.   // Show the times-up view.  
280.   switchSection(  
281.     $('#game-container'),  
282.     $('#times-up-container')  
283.   );  
284.  
285.   // Show the score.  
286.   $('#final-score').text(score);  
287. }  
-----Lines 288 through 295 Omitted-----
```

---

Evaluation  
Copy

## Conclusion

In this lesson, you have converted the Mathificent game from vanilla JavaScript to jQuery.

# LESSON 9

## Review of Mathificent jQuery Code

**EVALUATION COPY: Not to be used in class.**

### Topics Covered

- Understand the jQuery code behind Mathificent.

**Evaluation  
Copy**

### Introduction

In this lesson, we review the jQuery code used to create Mathificent.

### Conclusion

In this lesson, we have reviewed the jQuery code used to create Mathificent.