An Introduction to jQuery 16th Day

HTML, CSS, and JavaScript are the three fundamental languages of the web. We structure our websites with HTML, style them with CSS, and add interactive functionality with JavaScript. In fact, most animations and any action that happens as a result of a user clicking, hovering, or scrolling are constructed with JavaScript. JQuery is the “Write Less, Do More” JavaScript library. It is not a programming language, but rather a tool used to make writing common JavaScript tasks more concise. jQuery has the added benefit of being cross–browser compatible, meaning you can be certain the output of your code will render as intended in any modern browser.

By comparing a simple “Hello, World!” program in both JavaScript and jQuery, we can see the difference of how they’re both written.

JavaScript:

```
document.getElementById("demo").innerHTML = "Hello, World!";
```

Jquery:

```
$("#demo").html("Hello, World!");
```

Thie above example demonstrates how jQuery can achieve the same end result as plain JavaScript in a succinct manner.

Setting Up jQuery

jQuery is a JavaScript file that you will link to in your HTML. There are two ways to include jQuery in a project:

1). Download a copy of JQuery file.

2). Link to a file via Content Delivery Network (CDN).

Content Delivery Network (CDN) is a system of multiple servers that deliver web content to a user based on geographical location. When you link to a hosted jQuery file via CDN, it will potentially arrive faster and more efficiently to the user than if you hosted it on your own server.

We'll be using the CDN to reference jQuery in our examples. You can find the latest version of jQuery in the link below:

https://developers.google.com/speed/libraries/#jquery

Example: This is How we link Jquery CDN Link to our files.

```
<!doctype html>


<html lang="en">

<head>
  <title>jQuery Demo</title>
  <link rel="stylesheet" href="css/style.css">
</head>

<body>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
<script src="js/scripts.js"></script>
</body></html>
```

Note that:

- The CDN link should be added at the bottom like any other Js File.

- Your JavaScript file must be included *below* the jQuery library in the document or it will not work.

Using jQuery

At its core, jQuery is used to connect with HTML elements in the browser via the DOM.

The **Document Object Model** (DOM) is the method by which JavaScript (and jQuery) interact with the HTML in a browser. To view exactly what the DOM is, in your web browser, right click on the current web page select **Inspect**. This will open up Developer Tools. The HTML code you see here is the DOM.

Each HTML element is considered a **node** in the DOM – an object that JavaScript can touch. These objects are arranged in a tree structure, with `<html>` being closer to the root, and each nested element being a branch further along the tree. JavaScript can add, remove, and change any of these elements.

If you right click on the site again and click **View Page Source**, you will see the raw HTML output of the website. It's easy at first to confuse the DOM with the HTML source, but they're different – the page source is

exactly what is written in the HTML file. It is static and will not change, and will not be affected by JavaScript. The DOM is dynamic, and can change.

The outermost layer of the DOM, the layer that wraps the entire `<html>` node, is the **document** object. To begin manipulating the page with jQuery, we need to ensure the document is "ready" first.

Create the file `scripts.js` in your `js/` directory, and type the following code:

js/scripts.js:

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

// all custom jQuery will go here

});
```

All jQuery code you write will be wrapped in the above code. jQuery will detect this state of readiness so that code included inside this function will only run once the DOM is ready for JavaScript code to execute. Even if in some cases JavaScript won't be loaded until elements are rendered, including this block is considered to be best practice.

In the introduction of this article, you saw a simple "Hello, World!" script. To initiate this script and print text to the browser with jQuery, first we'll create an empty block–level paragraph element with the ID `demo` applied to it.

```
<body>

<p id="demo"></p>
```

jQuery is called with and represented by the dollar sign (`$`). We access the DOM with jQuery using mostly CSS syntax, and apply an action with a method. A basic jQuery example follows this format.

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

Since an ID is represented by a hash symbol (`#`) in CSS, we will access the demo ID with the selector `#demo`. `html()`is a method that changes the HTML within an element.

We're now going to put our custom "Hello, World!" program inside the jQuery `ready()` wrapper. Add this line to your `scripts.js` file within the existing function:

js/scripts.js

```
$(document).ready(function() {
    $("#demo").html("Hello, World!");
});
```

Once you've saved the file, you can open your `index.html` file in your browser. If everything worked properly, you will see the output `Hello, World!`.

If you were confused by the DOM before, you can see it in action now. Right click on the "Hello, World!" text on the page and choose **Inspect Element**. The DOM will now display `<p id="demo">Hello, World!</p>`. If you **View Page Source**, you will only see `<p id="demo"></p>`, the raw HTML we wrote.

Selectors.

Selectors are how we tell jQuery which elements we want to work on. Most

jQuery selectors are the same as what you're familiar with in CSS, with a

few jQuery–specific additions.

To access a selector, use the jQuery symbol `$`, followed by parentheses `()`.

```
$("selector")
```

It is advisable to use double-quoted strings though single-quoted strings are often used as well. Below is a brief overview of some of the most commonly used selectors.

- `$("*")` - **Wildcard**: selects every element on the page.
- `$(this)` - **Current**: selects the current element being operated on within a function.
- `$("p")` - **Tag**: selects every instance of the `<p>` tag.
- `$(".example")` - **Class**: selects every element that has the `example` class applied to it.
- `$("#example")` - **Id**: selects a single instance of the unique `example` Id,
- `$("[type='text']")` - **Attribute**: selects any element with `text` applied to the `type` attribute.
- `$("p:first-of-type")` - **Pseudo Element**: selects the first `<p>`.

Generally, classes and ids are what you will encounter the most — classes when you want to select multiple elements, and ids when you want to select only one.

jQuery Events

In the “Hello, World!” example, the code ran as soon as the page loaded and the document was ready, and therefore required no user interaction. In this case, we could have written the text directly into the HTML without bothering with jQuery. However, we will need to utilize jQuery if we want to make text appear on the page with the click of a button.

Return to your `index.html` file and add a `<button>` element. We will use this button to listen for our click event.

Index.html

```
...
<body>

<button id="trigger">Click me</button>
<p id="demo"></p>
```

We will use the `click()` method to call a function containing our “Hello, World!” code.

js/scripts.js

```
$(document).ready(function() {
    $("#trigger").click();
});
```

Our `<button>` element has an ID called `trigger`, which we select with `$("#trigger")`. By adding `click()`, we're telling it to listen for a click event, but we're not done yet. Now we'll invoke a function that contains our code, inside the `click()` method.

```
function() {
    $("#demo").html("Hello, World!");
}
```

Here's the final code.

js/scripts.js

```
$(document).ready(function() {
    $("#trigger").click(function() {
    $("#demo").html("Hello, World!");
     });
});
```

Save the `scripts.js` file, and refresh `index.html` in the browser. Now when you click the button, the "Hello, World!" text will appear.

An **event** is any time the user interacts with the browser. Usually this is done with the mouse or keyboard. The example we just created used a click event. (From the official jQuery documentation, you can view a full list of jQuery event methods.)

Below is a brief overview of some of the most commonly used event methods.

•`Click()`– **Click**: executes on a single mouse click.
•`Hover()` – **Hover**: executes when the mouse is hovered over an element. `Mouseenter()` and `mouseleave()` apply only to the mouse entering or leaving an element, respectively.
•`Submit()` – **Submit**: executes when a form is submitted.
•`scroll()` –**Scroll**: executes when the screen is scrolled.
•`Keydown()` – **Keydown**: executes when you press down on a key on the keyboard.
To make images animate or fade in as a user scrolls down the screen, use the `scroll()` method. To exit a menu using the `ESC` key, use the `keydown()` method. To make a dropdown accordion menu, use the `click()` method.

Understanding events is essential to creating dynamic website content with jQuery.

jQuery Effects

Jquery effects work hand–in–hand with events by allowing you to add animations and otherwise manipulate elements on the page.

We will make an example where we open and close a popup overlay. While we could use two IDs – one to open the overlay and another to close it – we'll instead use a class to open and close the overlay with the same function.

Delete the current `<button>` and `<p>` tags from within the body of your `index.html` file, and add the following to your HTML document:

index.html

```
...
<body>
<button class="trigger">Open</button>

<section class="overlay">
  <button class="trigger">Close</button>
</section>
...
```

In our `style.css` file, we will use a minimal amount of CSS to hide the `overlay` with `display: none` and center it on the screen.

css/style.css

```
.overlay {
  display: none;
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  height: 200px;
  width: 200px;
  background: gray;
}
```

Back in the `scripts.js` file, we're going to use the `toggle()` method, which will toggle the CSS `display` property between `none` and `block`, hiding and showing the overlay when clicked.

js/scripts.js

```
$(document).ready(function() {
    $(".trigger").click(function() {
        $(".overlay").toggle();
    });
});
```

Refresh `index.html`. You will now be able to toggle the visibility of the modal by clicking on the buttons. You can change `toggle()` to `fadeToggle()` or `slideToggle()` to see a few other built-in jQuery effects.

Below is a brief overview of some of the most commonly used effect

methods.

- `Toggle()` – **Toggle**: switches the visibility of an element or elements. `show()` and `hide()` are the related one-way effects.
- `FadeToggle()` – **Fade Toggle**: switches the visibility and animates the opacity of an element or elements. `FadeIn()` and `fadeOut()` are the related one-way effects.
- `SlideToggle()` – **Slide Toggle** toggles the visibility of an element or elements with a sliding effect. `slideDown()` and `slideUp()` are the related one-way effects.
- `Animate()` – **Animate** performs custom animation effects on the CSS property of an element.

We use jQuery events to listen for a user interaction such as the click of a

button, and we use jQuery effects to further manipulate elements once that

action takes place.

we learned how to select and manipulate elements with jQuery, and how events and effects work together to make an interactive web experience for the user.