# Web Design and Programming

Week 12

11 July 2024

Instructor: Dr. Peeraya Sripian

# Course schedule

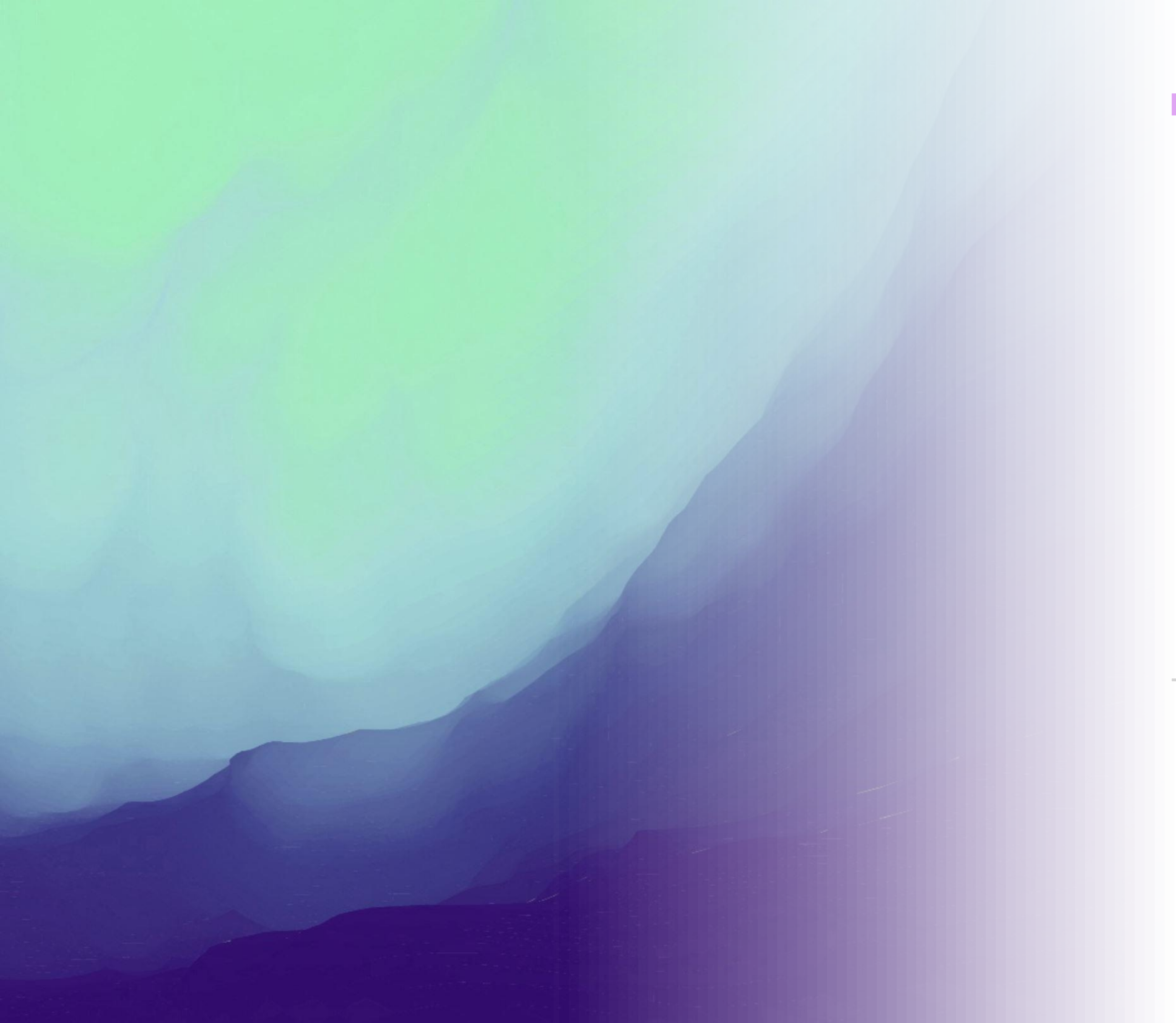
| Week | Date | Topic |
|---|---|---|
| 1 | 4/18 | Intro to WWW, Intro to HTML |
| 2 | 4/25 | CSS Fundamental |
|  | 5/2 | Holiday (GW) |
| 3 | 5/9 | CSS and Bootstrap |
| 4 | 5/16 | Work on midterm project |
| 5 | **COIL 22 MAY 18:00-19:30 (counted as 1 class, replacing 23 May)** |  |
| 6 | 5/30 | Midterm project presentation week |
| 7 | 6/6 | PHP fundamentals + Installation XAMPP |
| 8 | 6/13 | PHP fundamentals 2 + Intro of Final project |
| 9 | 6/20 | mySQL fundamentals |
| 10 | 6/27 | Assessing MySQL using PHP, MVC pattern |
| 11 | 7/4 | Cookies, sessions, and authentication **+ Proposal of final project** |
| 12 | 7/11 | Javascript ~~and PHP validation~~ |
| 13 | 7/18 | Final project development |
| 14 | 7/25 | Final project presentation |

## Today’s topic

- Javascript for behavior
- Using Javascript

Break 15 mins

- JQuery
- In Class Activity



# Javascript

For Behavior

## What Is JavaScript?

- JavaScript is a **client-side scripting language**—it is processed on the user's machine (not the server).
- It is reliant on the **browser's capabilities** (it may even be unavailable entirely).
- It is a **dynamic programming language**—it does not need to be compiled into an executable program. The browser reads it just as we do.
- It is **loosely** typed—you don't need to define variable types as you do for other programming languages.
- It has **nothing to do with Java**

## JavaScript Tasks

- JavaScript adds a **behavioral layer** (interactivity) to a web page. Some examples include:
- Checking **form submissions** and provide feedback messages and UI changes
- **Injecting** content into current documents on the fly
- **Showing** and **hiding** content based on a user clicking a link or heading
- Completing a term in a **search box**
- **Testing** for browser features and capabilities
- Much more!

## Some cool example website that use JavaScript

- Typewriting the code of the entire website
  - https://www.strml.net/
- Using Javascript to interact with the user in your webpage
  - http://www.histography.io/
- Hangman game
  - https://code.sololearn.com/WyyBylG1NvdU/#js
- Bouncing ball
  - https://codepen.io/b4rb4tron/pen/wjyXNJ

# Adding Scripts to a Page

- **Embedded script**
  Include the script in an HTML document with the **`script`** element:

```
<script>
  … JavaScript code goes here
</script>
```

- **External script**
  Use the `src` attribute in the **`script`** element to point to an external, standalone *.js* file:

```
<script src="my_script.js"></script>
```

# Script Placement

The `script` element can go anywhere in the document, but the most common places are as follows:

**In the `head` of the document**

For when you want the script to do something before the body completely loads (ex: Modernizr):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <script src="script.js"></script>
  </head>
  ...
```

**Just before the `</body>` tag**

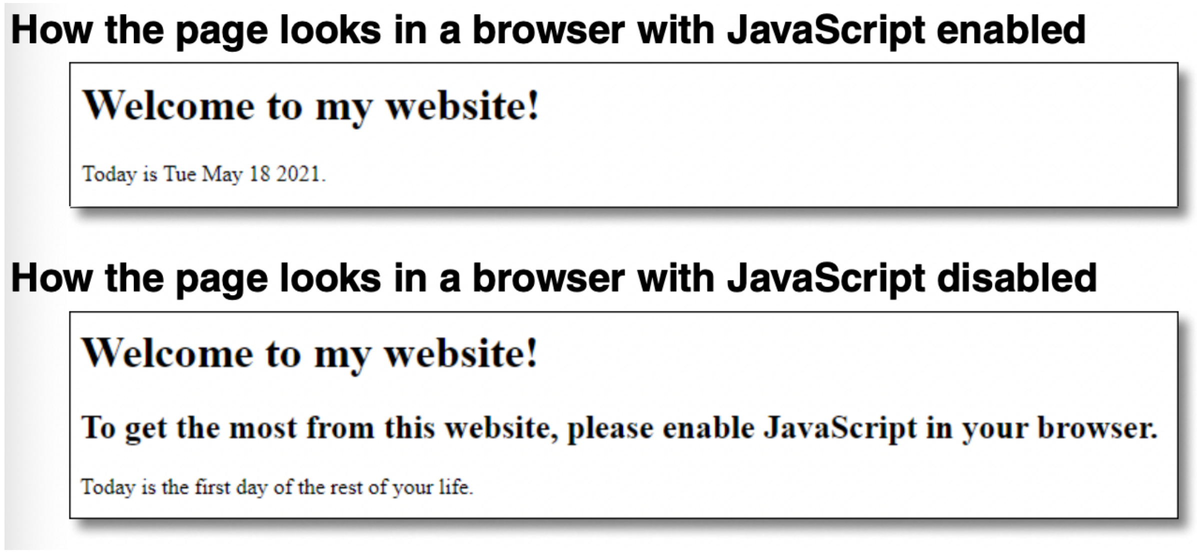
Preferred when the browser needs to parse the document and its DOM structure before running the script:

```
...
<body>
  <!--contents of page-->
<script src="script.js"></script>
</body>
</html>
```

## The noscript element

- If a user has JavaScript disabled in their browser -> JavaScript code won't run -> your website won't function
- Noscript element can be used to include the content that you want to display when a user has JavaScript disabled in their browser

```
<body>
  <header>
    <h1>Welcome to my website!</h1>
    <noscript>
      <h2>To get the most from this website,
      please enable JavaScript in your
      browser.</h2>
    </noscript>
    </header>
<!--main HTML for page goes here -->
  <footer>
    <script> const today = new Date();
      document.write(`Today is
      ${today.toDateString()}.`);
    </script>
    <noscript>
      Today is the first day of the rest of your
      life.
    </noscript>
  </footer>
</body>
```


**How the page looks in a browser with JavaScript enabled**

**Welcome to my website!**

Today is Tue May 18 2021.

**How the page looks in a browser with JavaScript disabled**

**Welcome to my website!**

**To get the most from this website, please enable JavaScript in your browser.**

Today is the first day of the rest of your life.

# JavaScript Syntax Basics

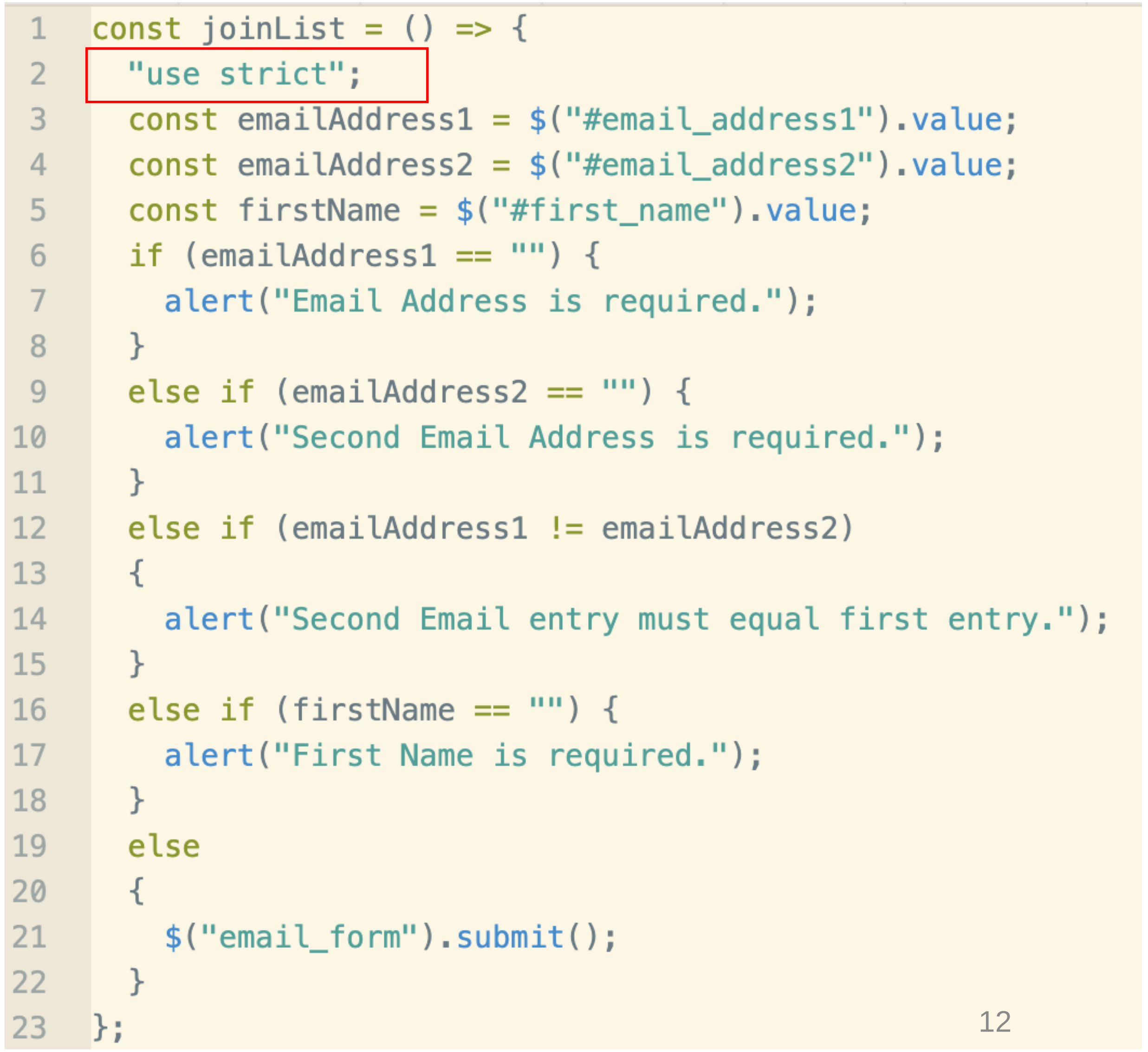
- JavaScript is **case-sensitive**.
- **Whitespace is ignored** (unless it is enclosed in quotes in a text string).
- A script is made up of a series of **statements**, commands that tell the browser what to do.
- **Single-line comments** in JavaScript appear after two `//` characters:

```
// This is a single-line comment
```

- **Multiple-line comments** go between `/*` and `*/` characters.

# Write a safer and cleaner code

- In non-strict mode, JavaScript also allows you to declare a variable without using a keyword.
  - However, doing so leads to some unexpected behavior, which can lead to bugs that are hard to track down.
- In strict mode, though, if you try to declare a variable without a keyword, JavaScript throws an error.
  - This alerts you to problems right away and helps you write safer code.
  - To enable strict mode, you code the “use strict” directive at the top of a code file or at the top of a function.



```
const joinList = () => {
  "use strict";
  const emailAddress1 = $("#email_address1").value;
  const emailAddress2 = $("#email_address2").value;
  const firstName = $("#first_name").value;
  if (emailAddress1 == "") {
    alert("Email Address is required.");
  }
  else if (emailAddress2 == "") {
    alert("Second Email Address is required.");
  }
  else if (emailAddress1 != emailAddress2)
  {
    alert("Second Email entry must equal first entry.");
  }
  else if (firstName == "") {
    alert("First Name is required.");
  }
  else
  {
    $("email_form").submit();
  }
};
```

## Building Blocks of Scripts

- Variables
- Comparison operators
- `if/else` statements
- Loops
- Functions
- Scope

## Variables

- A **variable** is made up of a **name** and a **value**.
- You create a variable so that you can refer to the value by name later in the script.
- The value can be a number, text string, element in the DOM, or function, to name a few examples.
- Variables are defined using the `let` or `var` keyword (but `let` is more recommended):

```
let foo = 5;
```

  - The variable is named `foo`. The equals sign (`=`) indicates we are **assigning** it the numeric value of 5.

## Variables (cont’d)

- Rules for naming a variable:
  - It must start with a letter or underscore (`_`).
  - It may not contain character spaces. Use underscores or CamelCase instead.
  - It may not contain special characters `(! . , / ¥ + * =)`.
  - It should describe the information it contains.

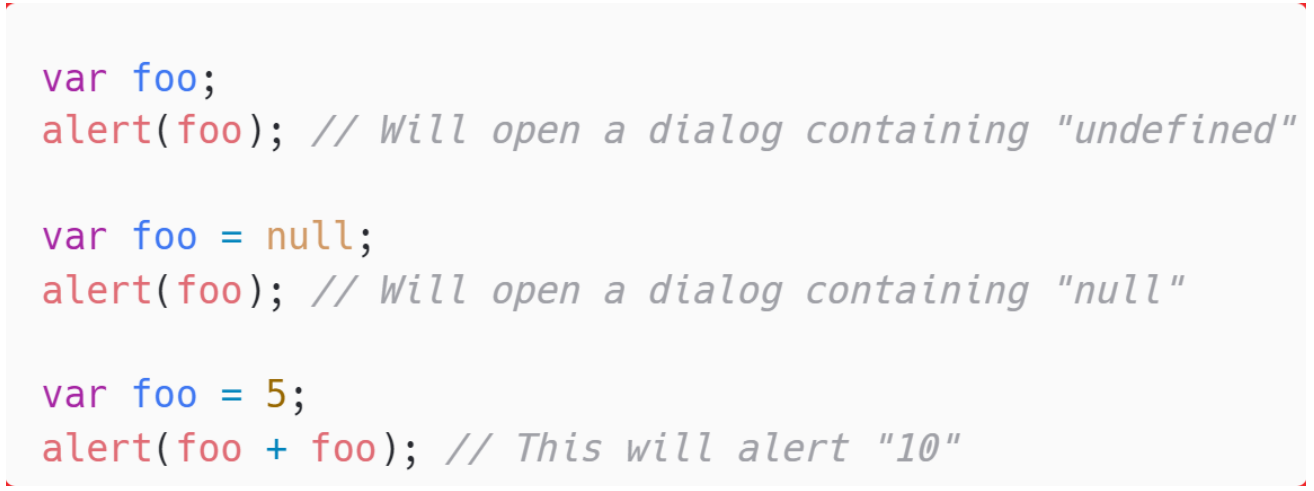
## Value Data Types

- Values assigned to variables fall under a few **data types**:

**Undefined**
The variable is declared by giving it a name, but no value:

**null**
Assigns the variable no inherent value:

**Numbers**
When you assign a number (e.g., 5), JavaScript treats it as a number (you don't need to tell it it's a number):



```
var foo;
alert(foo); // Will open a dialog containing "undefined"

var foo = null;
alert(foo); // Will open a dialog containing "null"

var foo = 5;
alert(foo + foo); // This will alert "10"
```

## Value Data Types (cont’d)

**Strings**
If the value is wrapped in single or double quotes, it is treated as a string of text:

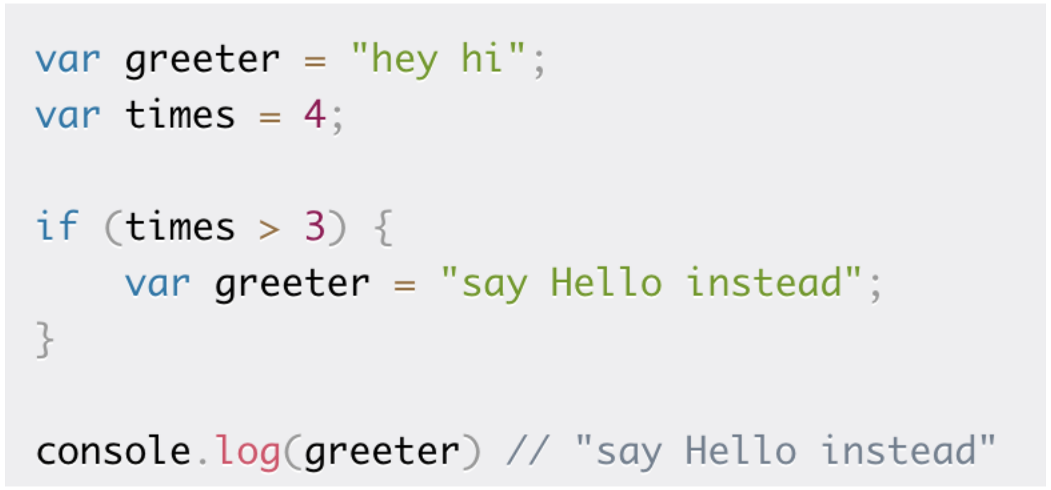
**Booleans**
Assigns a true or false value, used for scripting logic:

**Arrays**
A group of multiple values (called *members*) assigned to a single variable. Values in arrays are *indexed* (assigned a number starting with 0). You can refer to array values by their index numbers:

```
var foo = "five";
alert(foo); // Will alert "five"
alert(foo + foo); // Will alert "fivefive"

var foo = true; // The variable "foo" is now true

var foo = [5, "five", "5"];
alert( foo[0] ); // Alerts "5"
alert( foo[1] ); // Alerts "five"
alert( foo[2] ); // Also alerts "5"
```

## Problem with `var`

- Problem with `var` →
- This is fine only when you realize that a variable `greeter` has already been defined before
- However, this can cause confusion if you use `greeter` in other parts of your code



```
var greeter = "hey hi";
var times = 4;

if (times > 3) {
    var greeter = "say Hello instead";
}

console.log(greeter) // "say Hello instead"
```

# `let` is preferred for variable declaration

- `let` is the improvement to `var`
  - `let` can be updated but not re-declared
  - This will work

```
let greeting = "say Hi";
greeting = "say Hello instead";
```

This will return an error

```
let greeting = "say Hi";
let greeting = "say Hello instead";
```

  - This will be no error

Because both instances are treated as different variables since they have different scopes

```
let greeting = "say Hi";
if (true) {
    let greeting = "say Hello instead";
    console.log(greeting); // "say Hello instead"
}
console.log(greeting); // "say Hi"
```

## const

- Similar to `let` declarations
- `const` declaration are block scoped
- `const` cannot be updated or re-declared
- These will result in error →
- Every `const` declaration must be initialized at the time of declaration
- Property of the `const` can be updated

✗

```
const greeting = "say Hi";
greeting = "say Hello instead";
```

✗

```
const greeting = "say Hi";
const greeting = "say Hello instead";//
```

```
const greeting = {
    message: "say Hi",
    times: 4
}
```

→

○

```
greeting.message = "say Hello instead";
```

# Comparison Operators

- **Comparison operators** are special characters in JavaScript syntax that evaluate and compare values:

| Operators | Meaning |
|---|---|
| == | Is equal to |
| != | Is not equal to |
| === | Is identical to * |
| !== | Is not identical to * |
| > | Is greater than |
| >= | Is greater than or equal to |
| < | Is less than |
| <= | Is less than or equal to |

* Value is equal and of the same data type

**Example**

- JavaScript evaluates the statement and gives back a Boolean (true/false) value
- Equal to (==) is not the same as identical to (===). Identical values must share a data type

```javascript
alert( 5 == 5 ); // This will alert "true"
alert( 5 != 6 ); // This will alert "true"
alert( 5 < 1 );  // This will alert "false"

alert( "5" == 5 ); // This will alert "true". They're both "5".
alert( "5" === 5 ); // This will alert "false". They're both "5", but they're not the same data type.
alert( "5" !== 5 ); // This will alert "true", since they're not the same data type.
```

# Mathematical Operators

- **Mathematical operators** perform mathematical functions on numeric values:

| | |
|---|---|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Divide |
| += | Adds the value to itself |
| ++ | Increases the value of a number (or number in a variable) by 1 |
| -- | Decreases the value of a number (or number in a variable) by 1 |

## `if/else` Statements

- An `if/else` **statement** tests for conditions by asking a true/false question.
- **If** the condition in parentheses is met, then execute the commands between the curly brackets (`{}`):

```
if(true) {
  // Do something.
}
```

- **Example:**

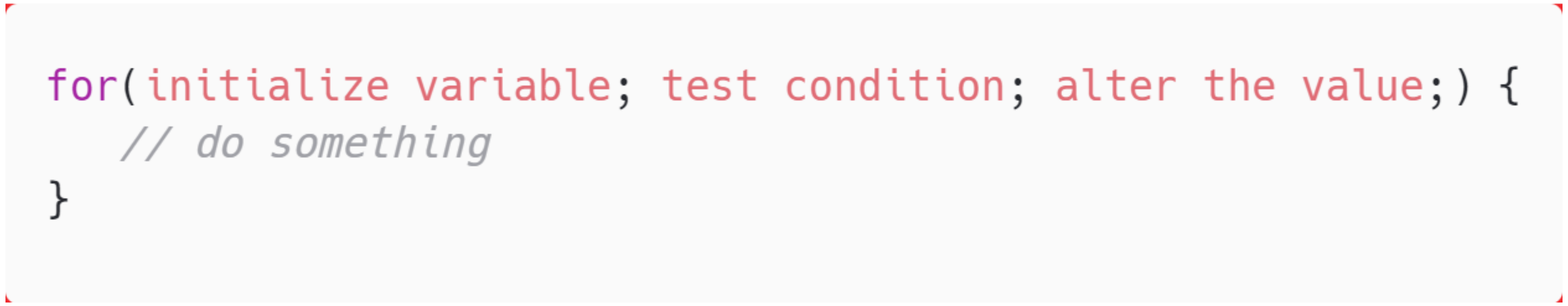
```
if( 1 != 2 ) {
  alert("These values are not equal.");
  // It is true that 1 is never equal to
2, so we should see this alert.
}
```

**also possible with else, Example:**

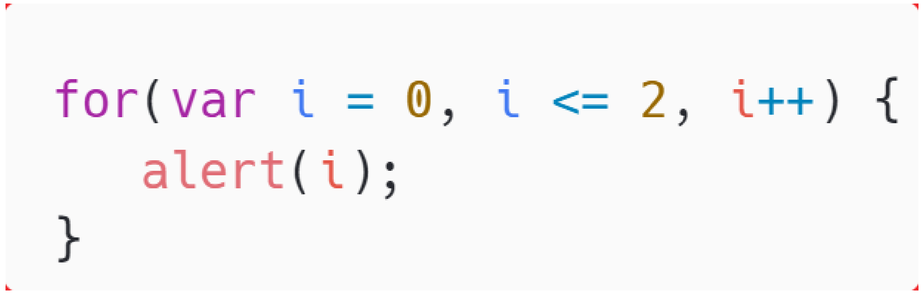
```
var test = "testing";
if( test == "testing" ) {
      alert( "You haven't changed
anything." );
} else {
      alert( "You've changed something!" );
}
```

# Loops

- **Loops** allow you to do something to every variable in an array without writing a statement for every one.
- One way to write a loop is with a `for` **statement**:



```
for(initialize variable; test condition; alter the value;) {
   // do something
}
```

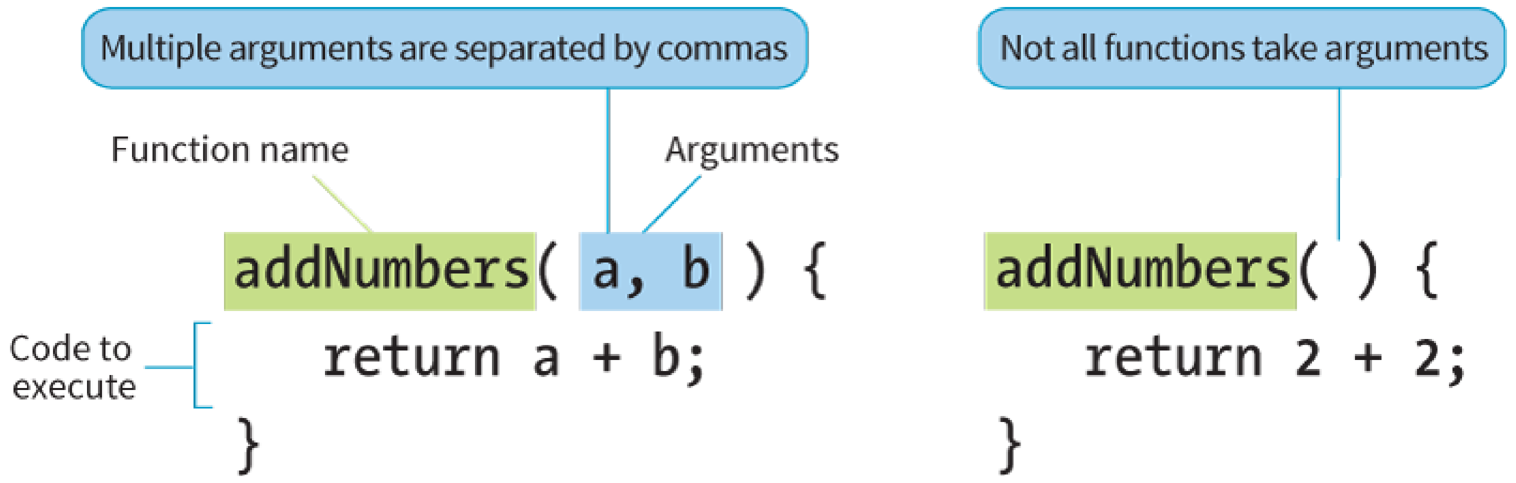


```
for(var i = 0, i <= 2, i++) {
    alert(i);
}
```

Trigger 3 alerts
Reading 0, 1, 2

# Functions

- A **function** is a bit of code for performing a task that doesn't run until it is referenced or called.
- Parentheses sometimes contain **arguments** (additional information used by the function):


Multiple arguments are separated by commas
Not all functions take arguments
Function name
Arguments
Code to execute

```
addNumbers( a, b ) {
    return a + b;
}
```

```
addNumbers( ) {
    return 2 + 2;
}
```

## Functions (cont'd)

- Some functions are built into JavaScript. Here are examples of **native functions**:
  - `alert()`, `confirm()`, and `prompt()`
    Functions that trigger browser-level dialog boxes
  - `date()`
    Returns the current date and time
- You can also create your own **custom functions** by typing `function` followed by a name for the function and the task it performs:

```
function name() {
  // Code for the new function goes here.
}
```

# Variable Scope

- A variable that can only be used within one function is **locally scoped**. When you define a variable inside a function, include the **`var`** keyword to keep it locally scoped (recommended):

```
var foo = "value";
```

- A variable that can be used by any script on your page is said to be **globally scoped**.
  - Any variable created *outside* of a function is automatically globally scoped:

```
var foo = "value";
```

  - To make a variable created *inside* a function globally scoped, omit the **`var`** keyword:

```
foo = "value";
```

# The Browser Object

- JavaScript lets you manipulate parts of the browser window itself (the `window` object).
- Examples of `window` properties and methods:

| **Property/Method** | **Description** |
|---|---|
| `event` | Represents the state of an event |
| `history` | Contains the URLs the user has visited within a browser window |
| `location` | Gives read/write access to the URI in the address bar |
| `status` | Sets or returns the text in the status bar of the window |
| `alert()` | Displays an alert box with a specified message and an OK button |
| `close()` | Closes the current window |
| `confirm()` | Displays a dialog box with a specified message and an OK and a Cancel button |
| `focus()` | Sets focus on the current window |

## Event Handlers

- An **event** is an action that can be detected with JavaScript and used to trigger scripts.
- Events are identified by **event handlers**. Examples:
  - `onload` When the page loads
  - `onclick` When the mouse clicks an object
  - `onmouseover` When the pointer is moved over an element
  - `onerror` When an error occurs when the document or a resource loads

## Event Handlers (cont’d)

- Event handlers can be applied to items in pages in three ways:
  - As an HTML attribute:

```
<body onclick="myFunction();">
/* myFunction runs when the user clicks anything within 'body' */
```

  - As a method attached to the element:

```
window.onclick = myFunction;
/* myFunction will run when the user clicks anything within the browser window */
```

  - Using **`addEventListener()`**:

```
window.addEventListener("click", myFunction);
```

Notice that we omit the preceding “on” from the event handler with this syntax.

## Debugging code

https://developer.chrome.com/docs/devtools/javascript/

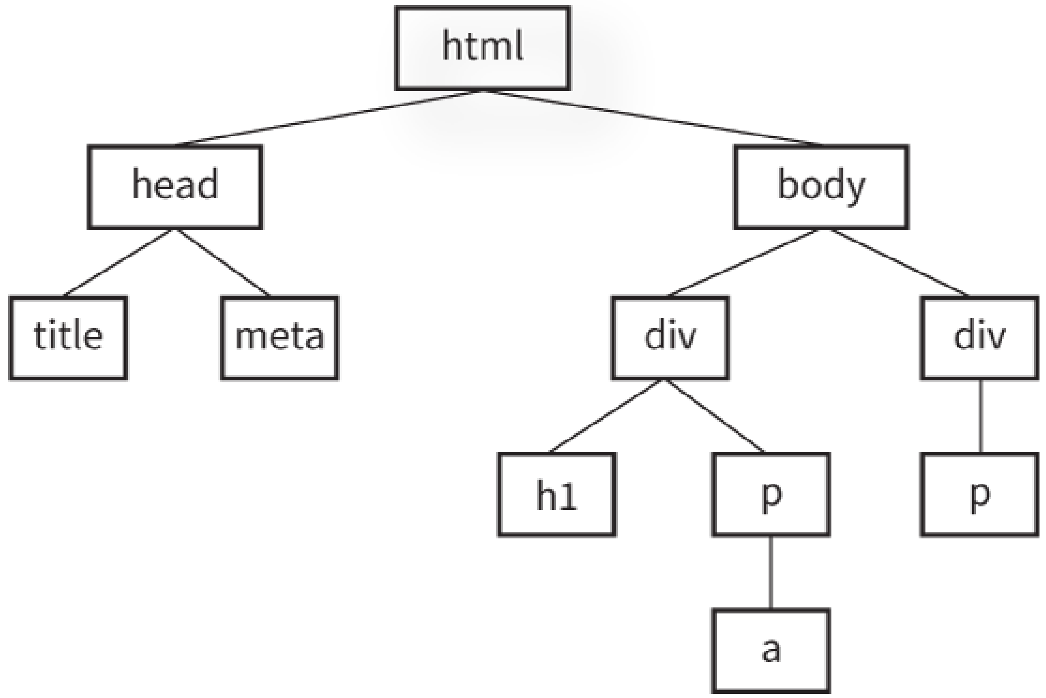
# Using Javascript

## Intro to the DOM

- The **Document Object Model (DOM)** is a **programming interface** that provides a way to access and manipulate the contents of a document.
- It provides a structured **map of the document** and a set of **methods** for interacting with them.
- It can be used with other XML languages and it can be accessed by other programming languages (like PHP, Ruby, etc.).

## Node Tree

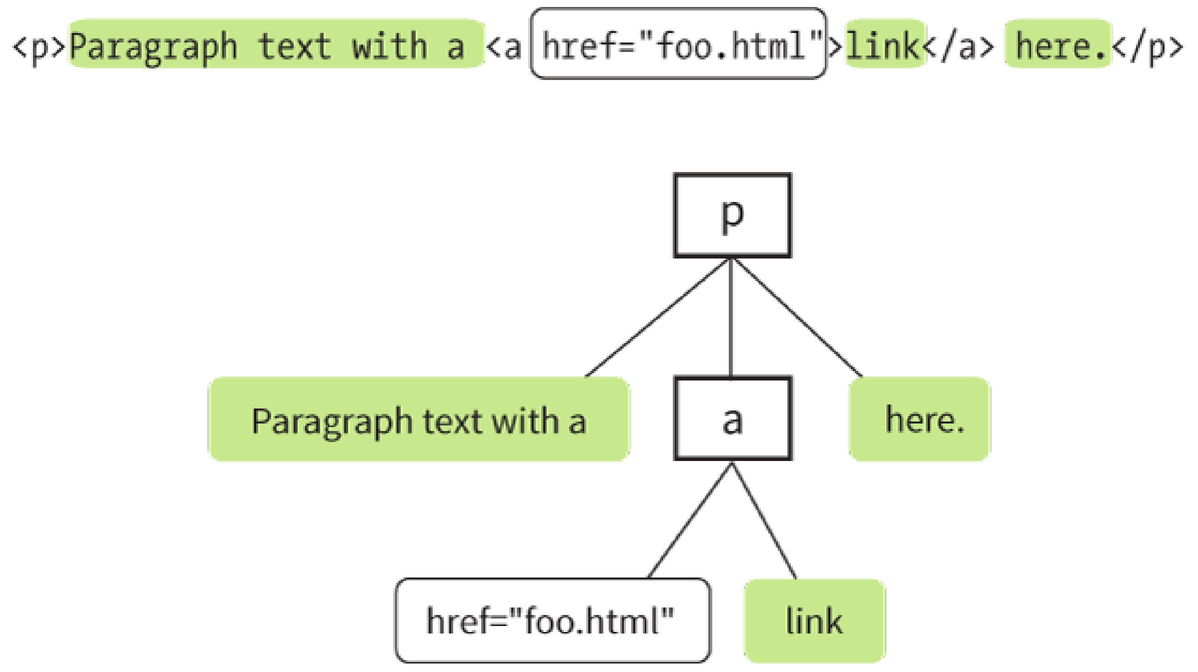
- The DOM treats the structure of a document like a tree with branches:



## Node Tree (cont'd)

- Every element, attribute, and piece of content is a node on the tree and can be accessed for scripting:

The nodes within a `p` element



```
<p>Paragraph text with a <a href="foo.html">link</a> here.</p>
```

## Accessing Nodes

- To point to nodes, list them separated by periods (.).
- In this example, the variable `foo` is set to the HTML content of an element with `id="beginner"`:

```
var foo = document.getElementById("beginner").innerHTML;
```

  - The **`document`** object points to the page itself.
  - **`getElementById`** specifies an element with the `id` “beginner”.
  - **`innerHTML`** stands for the HTML content within that element.

## Accessing Nodes (cont’d)

Methods for accessing nodes in the document:

`getElementsByTagName()`

Accesses all elements with the given tag name

Example:

```
document.getElementsByTagName("p");
```

`getElementById()`

Accesses a single element by the value of its id attribute

Example:

```
document.getElementById("special");
```

`getElementsByClassName()`

Access elements by the value of a class attribute

Example:

```
document.getElementsByClassName("product");
```

`querySelectorAll()`

Accesses nodes based on a CSS selector

Example:

```
document.querySelectorAll(".sidebar p");
```

`getAttribute()`

Accesses the value of a given attribute

Example:

```
getAttribute("src")
```

# Manipulating Nodes

- There are several built-in methods for manipulating nodes:

`setAttribute()`

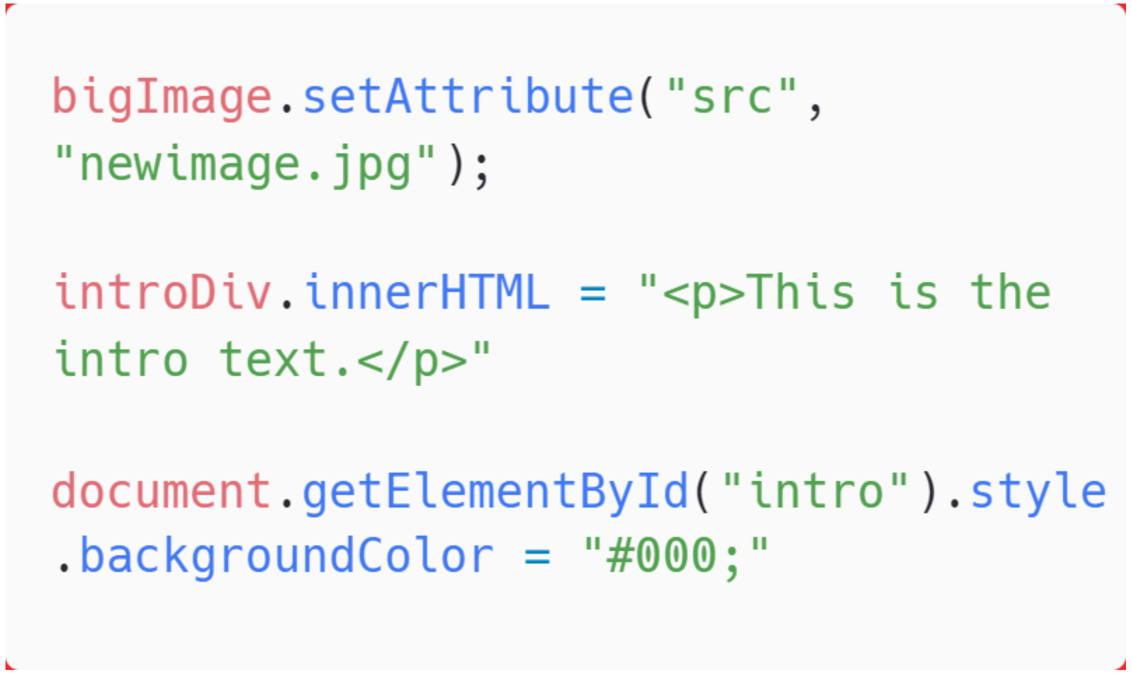
Sets the value of a given attribute:

`innerHTML`

Specifies the content inside an element (including markup if needed):

`style`

Applies a style using CSS properties:



```
bigImage.setAttribute("src",
"newimage.jpg");

introDiv.innerHTML = "<p>This is the
intro text.</p>"

document.getElementById("intro").style
.backgroundColor = "#000;"
```

## Adding and Removing Elements

- The DOM allows developers to change the document structure by adding and removing nodes:

```
createElement()
createTextNode()
appendChild()
insertBefore()
replaceChild()
removeChild()
```

## Polyfills

- A `polyfill` uses JavaScript to make new features work in browsers that don’t natively support them.
- `Picturefill`: Enables support for `picture`, `srcset`, and `sizes`
- `Selectivizr*`: Allows IE 6–8 to support CSS3 selectors
- `HTML5 shiv*`: Allows IE6–8 to recognize HTML5 elements

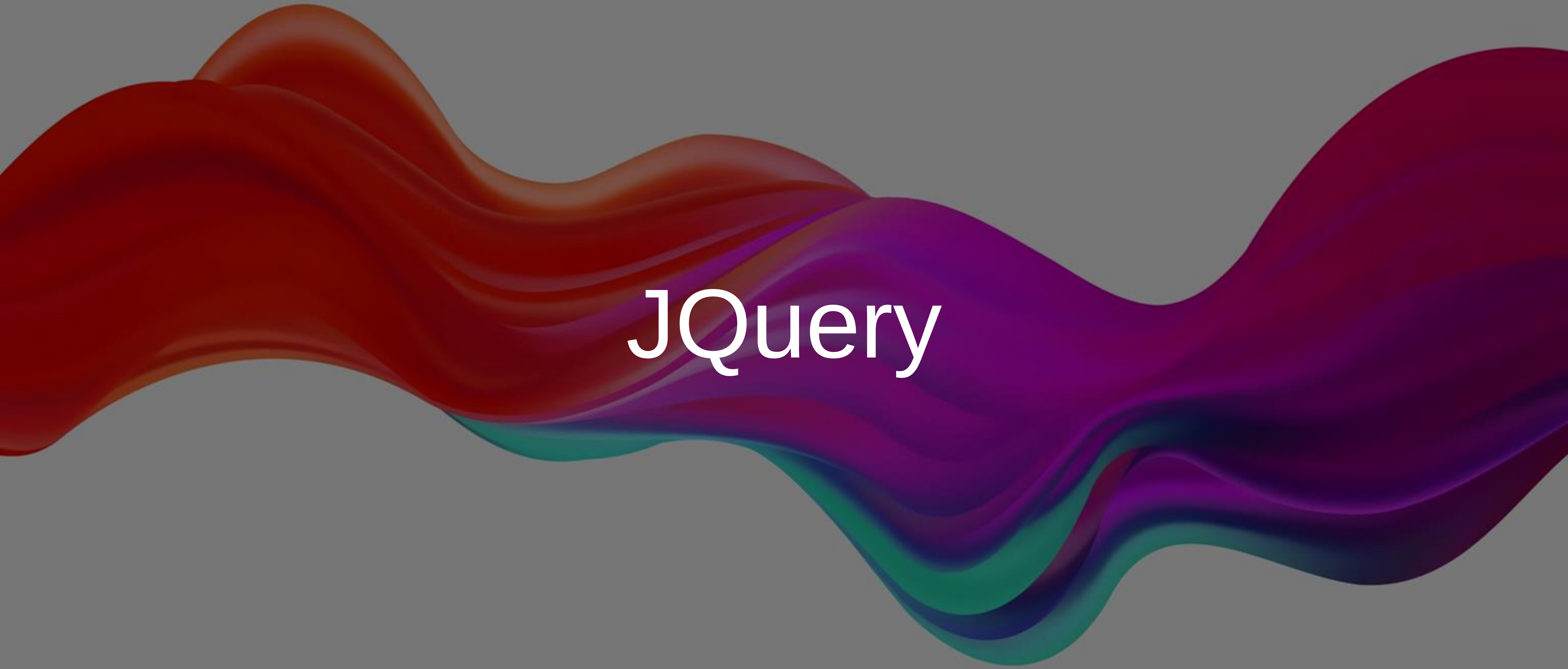
*If you don’t need to support IE 8 and earlier, you don’t need these polyfills.

# JavaScript Libraries

- A **JavaScript library** is a collection of prewritten functions and methods that you can use in your scripts to accomplish common tasks or simplify complex ones.
- Some are large frameworks for building complex applications.
- Some are targeted to specific tasks, such as forms or math.
- The most popular library is **jQuery**.
- Try searching “JavaScript library for ___________” to see if there are pre-made scripts you can use or adapt to your needs.

# 15 minutes break

Class resume at 10:25 AM



## JQuery

## What is jQuery?

- jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development
- Free and open JavaScript Library
- Work across modern browsers
- Abstracts away browser-specific features, allowing you to concentrate on design

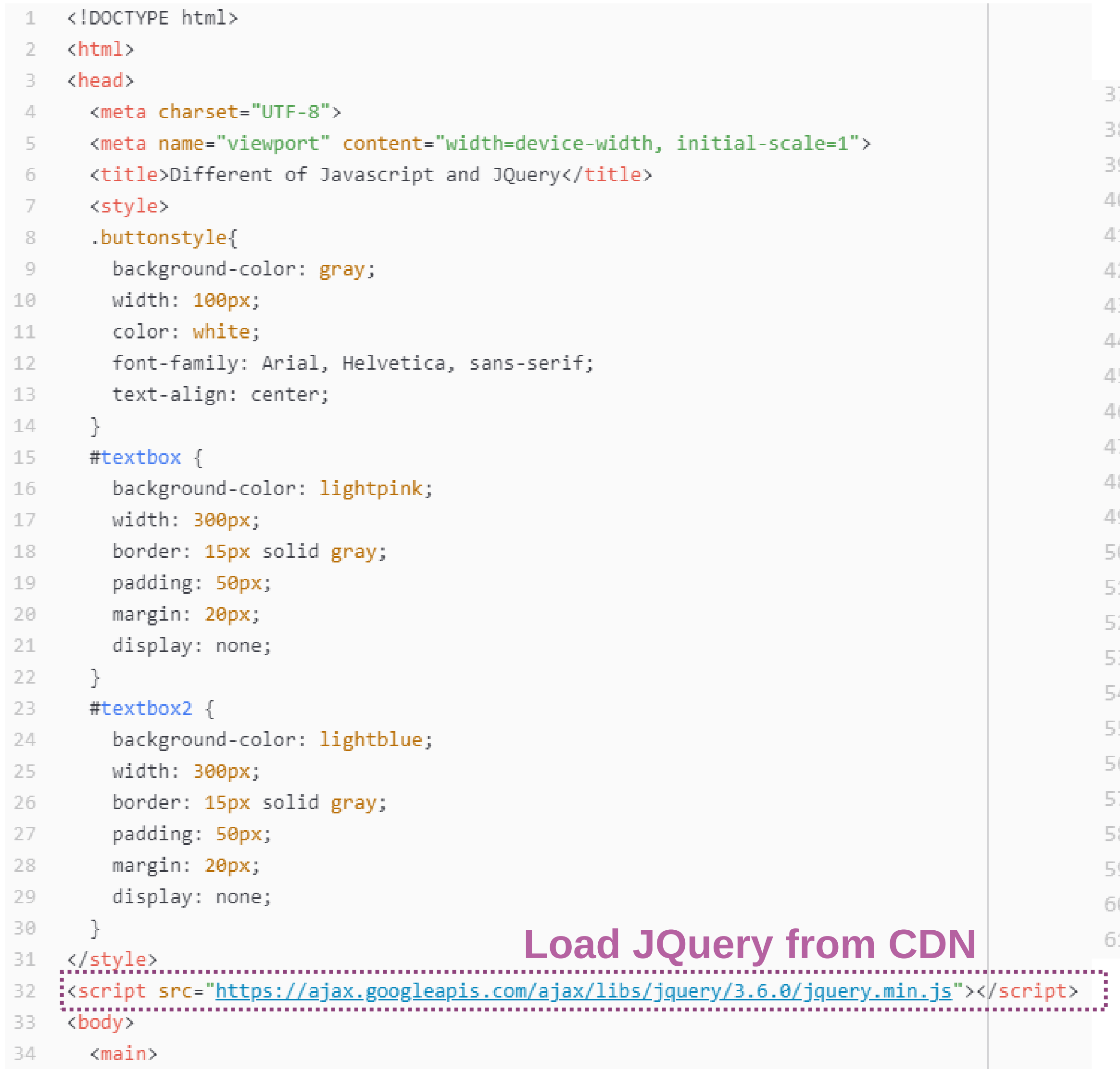
# Why learn jQuery?

- Write less, do more:

```
$("p.neat").addClass("ohmy").show("slow");
```

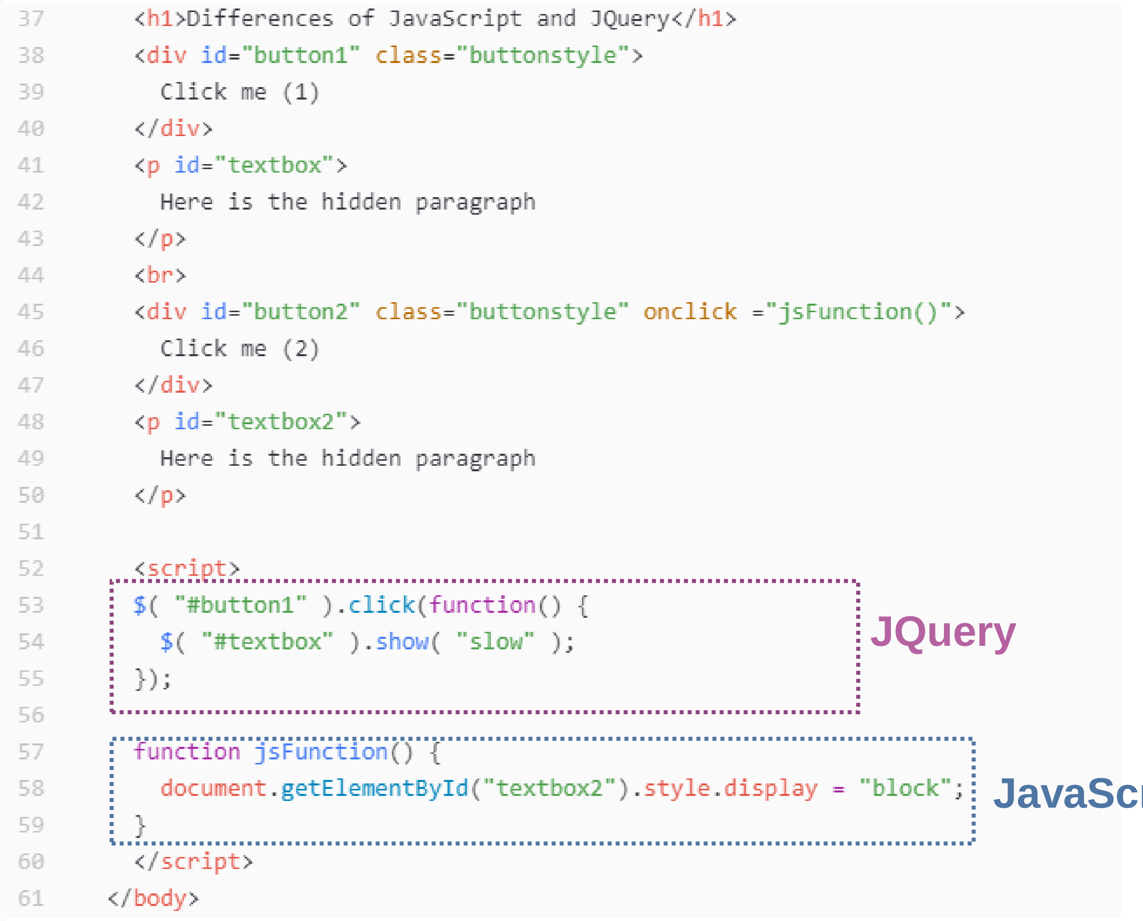
- Performance
- Plugins
- It’s standard
- … and fun!

# Example: Click me to show something



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Different of Javascript and JQuery</title>
  <style>
  .buttonstyle{
    background-color: gray;
    width: 100px;
    color: white;
    font-family: Arial, Helvetica, sans-serif;
    text-align: center;
  }
  #textbox {
    background-color: lightpink;
    width: 300px;
    border: 15px solid gray;
    padding: 50px;
    margin: 20px;
    display: none;
  }
  #textbox2 {
    background-color: lightblue;
    width: 300px;
    border: 15px solid gray;
    padding: 50px;
    margin: 20px;
    display: none;
  }
</style>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<body>
  <main>
```

Load JQuery from CDN



```
    <h1>Differences of JavaScript and JQuery</h1>
    <div id="button1" class="buttonstyle">
      Click me (1)
    </div>
    <p id="textbox">
      Here is the hidden paragraph
    </p>
    <br>
    <div id="button2" class="buttonstyle" onclick ="jsFunction()">
      Click me (2)
    </div>
    <p id="textbox2">
      Here is the hidden paragraph
    </p>

    <script>
    $( "#button1" ).click(function() {
      $( "#textbox" ).show( "slow" );
    });

    function jsFunction() {
      document.getElementById("textbox2").style.display = "block";
    }
    </script>
  </body>
```

JQuery

JavaScript

## jQuery terminology

- the jQuery function

  refers to the global jQuery object or the $ function depending on the context

- a jQuery object

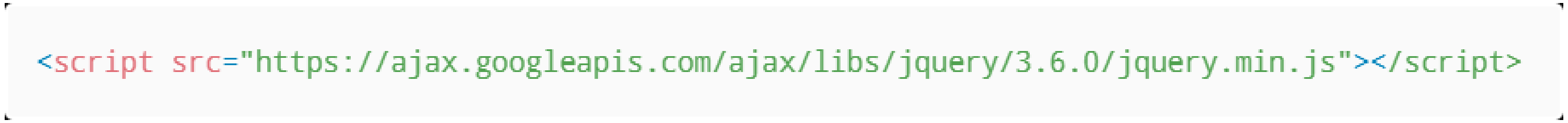
  the object returned by the jQuery function that often represents a group of elements

- selected elements

  the DOM elements that you have selected for, most likely by some CSS selector passed to the jQuery function and possibly later filtered further

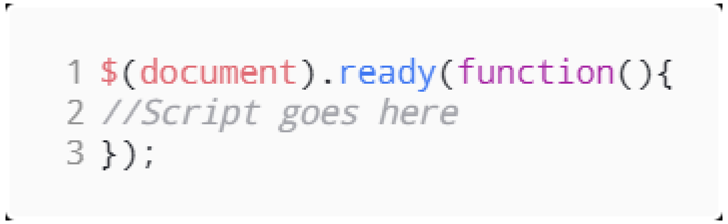
# Enable jQuery in your webpage

- jQuery can be enabled in your page by including reference to jQuery library file
  - Get the CDN from here:
    - https://developers.google.com/speed/libraries#jquery
    - https://releases.jquery.com/



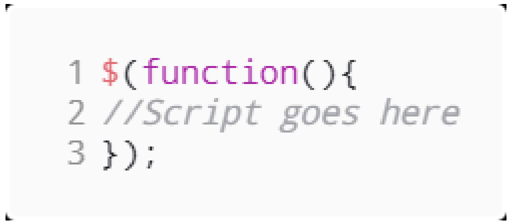
```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

- Introduce a jQuery function by using the below given function.



```
$(document).ready(function(){
//Script goes here
});
```

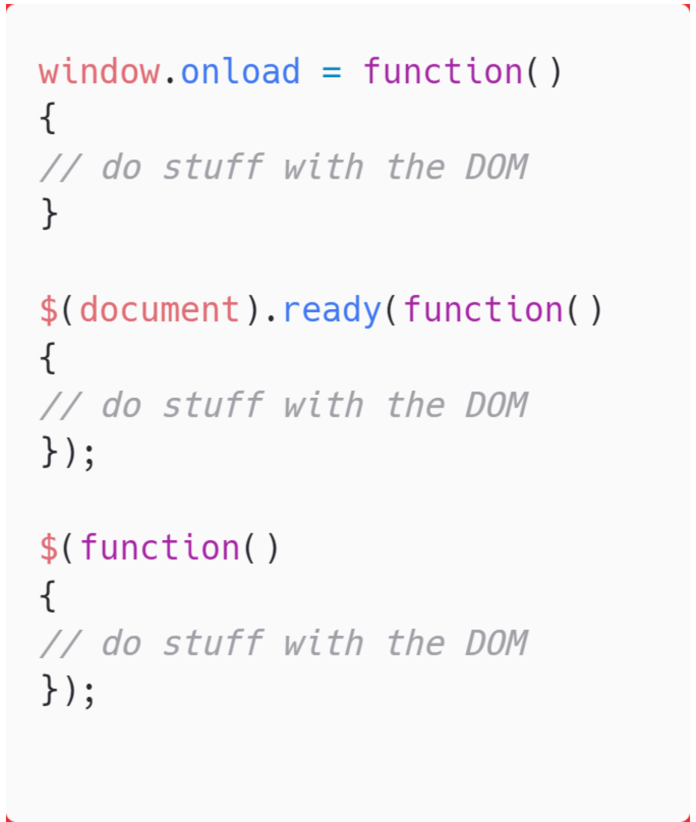
OR



```
$(function(){
//Script goes here
});
```

## `window.onload`

- We cannot use the DOM before the page has been constructed. jQuery gives us a more compatible way to do this.
  - The DOM way
  - The direct jQuery translation
  - The jQuery way

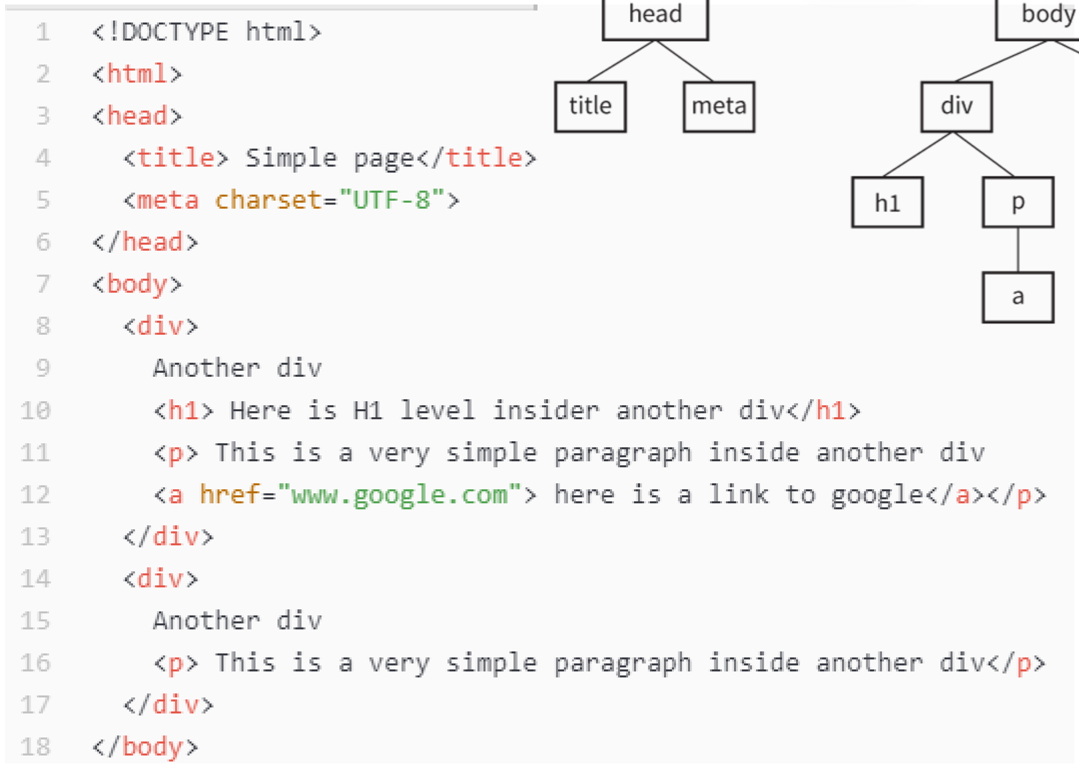


```
window.onload = function()
{
// do stuff with the DOM
}

$(document).ready(function()
{
// do stuff with the DOM
});

$(function()
{
// do stuff with the DOM
});
```

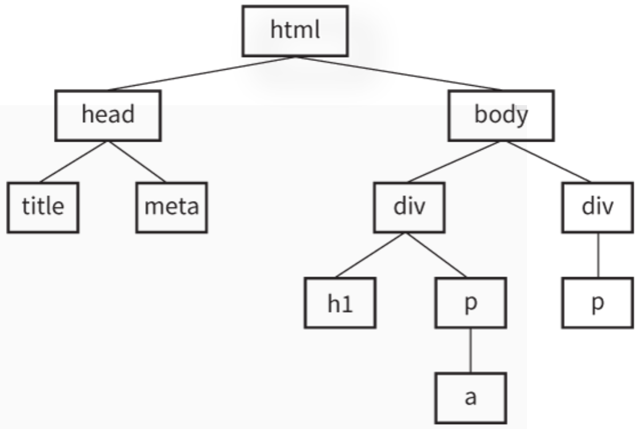
## Aspects of the DOM and jQuery

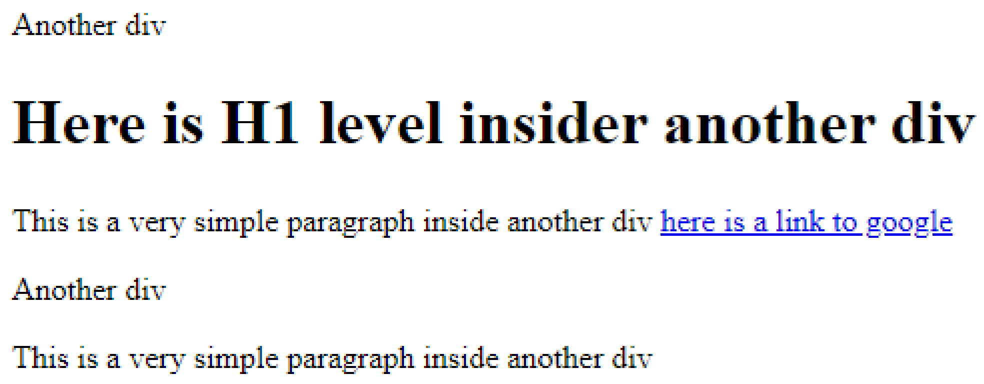
- **Identification:** how do I obtain a reference to the node that I want.
- **Traversal:** how do I move around the DOM tree.
- **Node Manipulation:** how do I get or set aspects of a DOM node.
- **Tree Manipulation:** how do I change the structure of the page.

# The DOM tree again



```
<!DOCTYPE html>
<html>
<head>
  <title> Simple page</title>
  <meta charset="UTF-8">
</head>
<body>
  <div>
    Another div
    <h1> Here is H1 level insider another div</h1>
    <p> This is a very simple paragraph inside another div
    <a href="www.google.com"> here is a link to google</a></p>
  </div>
  <div>
    Another div
    <p> This is a very simple paragraph inside another div</p>
  </div>
</body>
```





Another div

**Here is H1 level insider another div**

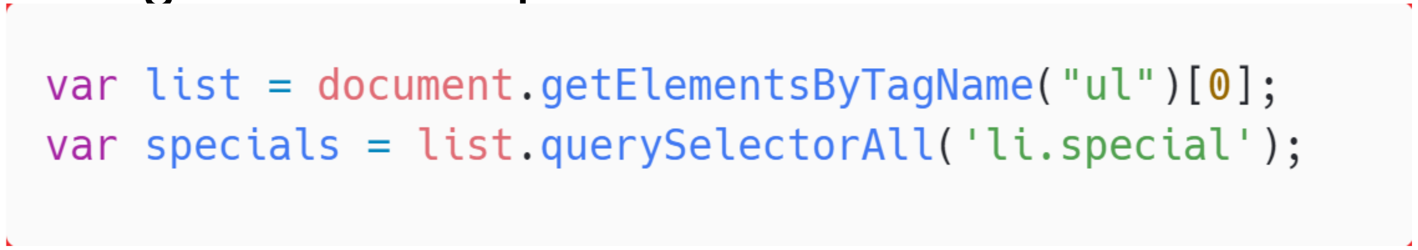
This is a very simple paragraph inside another div here is a link to google

Another div

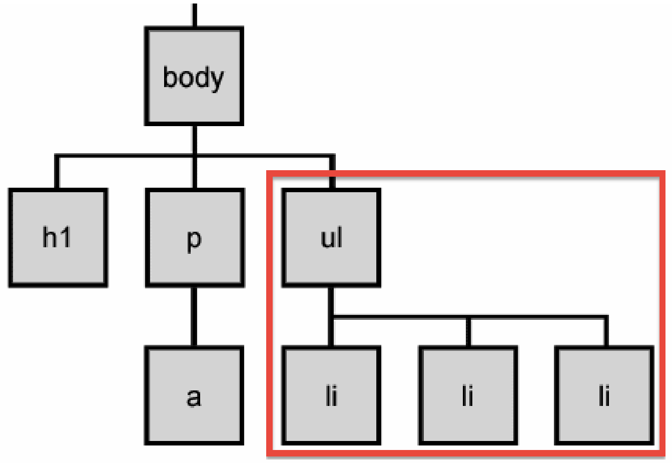
This is a very simple paragraph inside another div

## DOM context identification

- You can use `querySelectorAll()` and `querySelector()` on any DOM object.
- When you do this, it simply searches from that part of the DOM tree downward.
- Programmatic equivalent of a CSS context selector



```
var list = document.getElementsByTagName("ul")[0];
var specials = list.querySelectorAll('li.special');
```

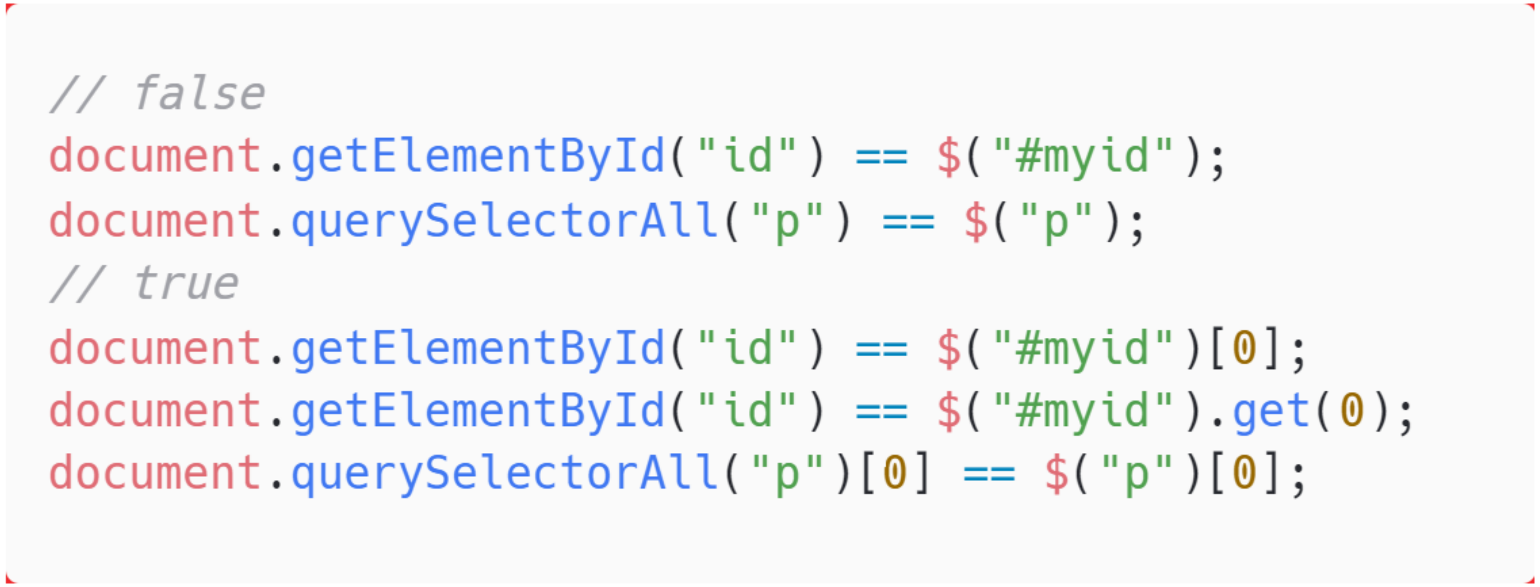


## jQuery Selectors

- jQuery borrows from CSS, utilizing the **selectors**, as well as adding its own, which are used for matching a set of **elements** from the HTML DOM.
- Selectors in jQuery are meant to specify **a set of elements** based on certain attributes, such as ID, class, or the type of tag itself.
- These elements can then be **selected** for applying the jQuery method or a function you define.

# The jQuery object

- The $ function always (even for ID selectors) returns an array-like object called a jQuery object.
- The jQuery object wraps the originally selected DOM objects.
- You can access the actual DOM object by accessing the elements of the jQuery object.



```javascript
// false
document.getElementById("id") == $("#myid");
document.querySelectorAll("p") == $("p");
// true
document.getElementById("id") == $("#myid")[0];
document.getElementById("id") == $("#myid").get(0);
document.querySelectorAll("p")[0] == $("p")[0];
```

## Using $ as a wrapper

- **$** adds extra functionality to DOM elements
- passing an existing DOM object to **$** will give it the jQuery upgrade

```
// convert regular DOM objects to a jQuery object
var elem = document.getElementById("myelem");
elem = $(elem);
var elems = document.querySelectorAll(".special");
elems = $(elems);
```

## jQuery Selectors

- All selectors in jQuery start with `$`
- Example: `$("button")`
  - select every element with the **<button>** tag in the document
- **Caution:**
  - Although many of the meta characters are used as selectors, you can include them in the values of class and ID attributes when selecting as well.
  - However, they **must be escaped** using to **backslashes** before the character.
  - For example, if you wanted to select an element with the **id** attribute with the value **nav.bar**, the selector would be `$("nav//.bar")` and not `$("nav.bar")`

## Combining Selectors

- Possible to combine multiple selectors in jQuery

```
$("selector1, selector2, selectors3, ..., selectorn")
```

- When selecting multiple **attributes**, however, you do not need to use the commas to separate them. You may simply place the jQuery attribute selectors one after another:

```
$("[attribute1='value'][attribute2='value'][attribute3='value']")
```

# jQuery node identification (basic selectors)

```
// id selector (jQuery)
let elem = $("#myid");

// group selector
var elems = $("#myid, p");


// context selector
var elems = $("#myid < div p");

// complex selector
var elems = $("#myid < h1.special:not(.classy)");
```

In Javascript

```
// id selector (JavaScript)
let elem = document.getElementById(“id”)

// group selector is not exist in JavaScript
var x = document.getElementById(“myid”);
var y = x.getElementsByTagName(“p”);
```

# Selecting by Attribute Value

- jQuery has a plenty of selectors that select HTML elements with **attributes** that meet certain conditions.
- They are commonly referred to as jQuery attribute selectors.

`$("[attribute]")` - selects a set of elements that have the specified attribute.

`$("[attribute='value']")` - selects a set of elements that have the specified attribute with the specified value.

`$("[attribute!='value']")` - selects a set of elements that **do not** have the specified attribute with the specified value.

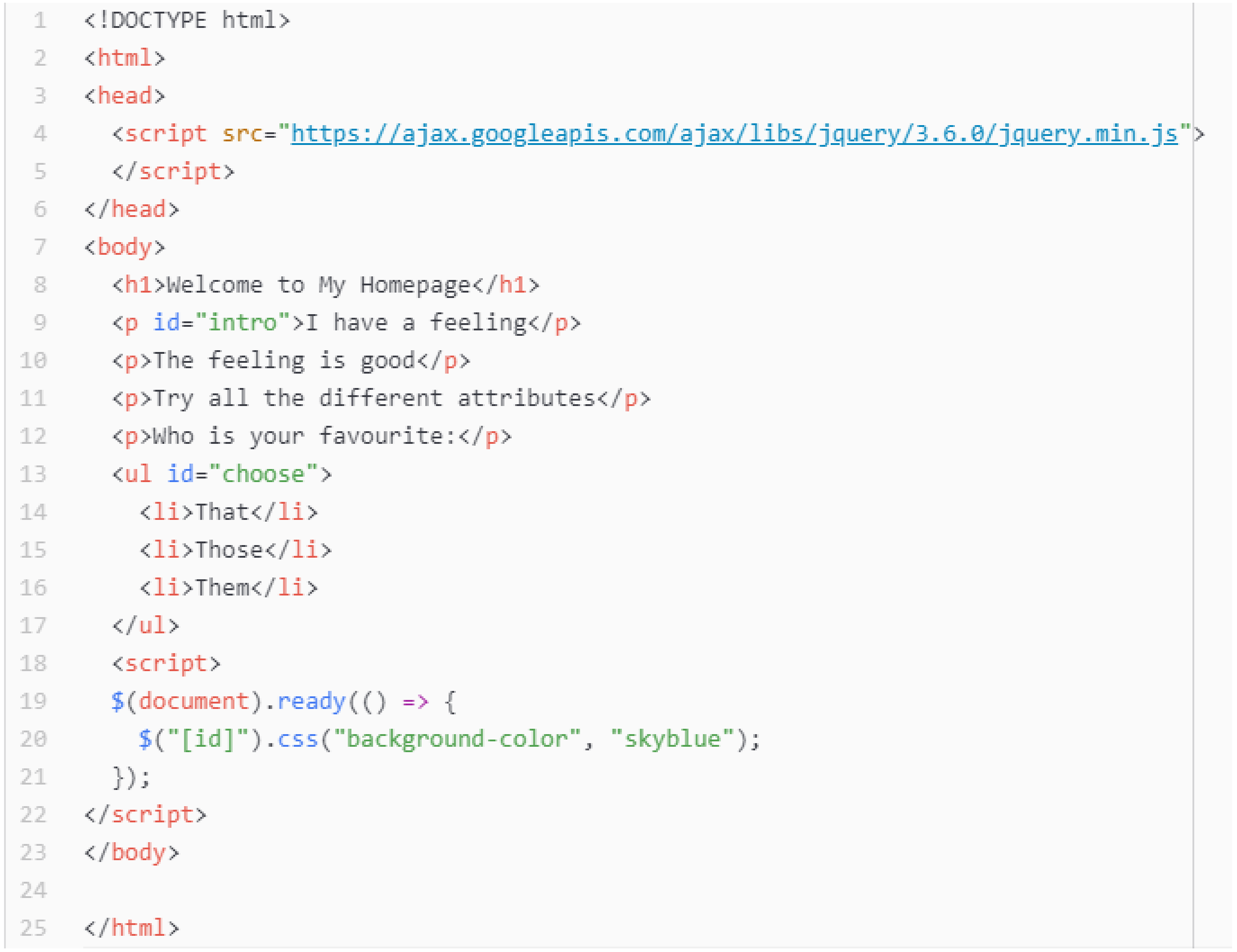
`$("[attribute|='prefix']")` - selects a set of elements that have the specified attribute with the value that has a specified prefix (separated from the rest of the value name by a hyphen).

`$("[attribute*='value']")` - selects a set of elements that have the specified attribute with the value that contains the specified substring. A substring can be a part of another string anywhere inside it, so the phrase you specify as the value **does not** have match the whole.

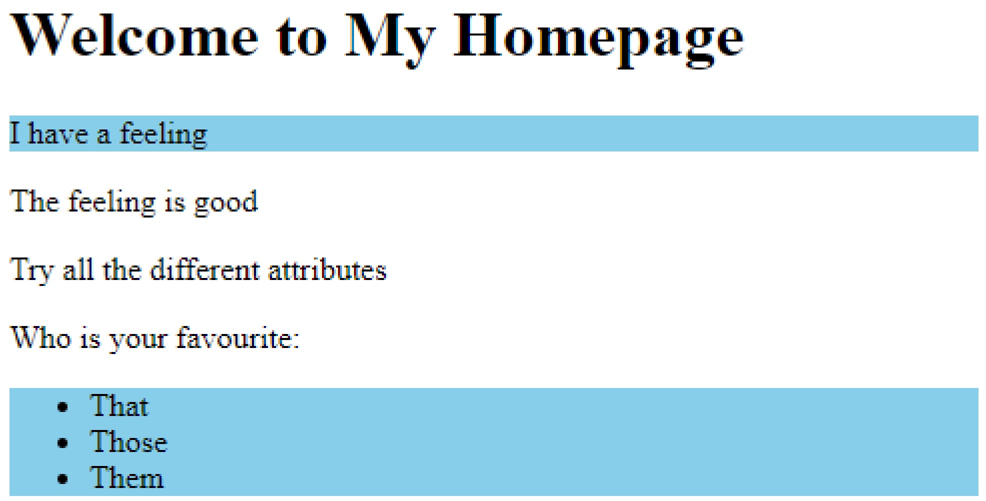
`$("[attribute$='value']")` - selects a set of elements that have the specified attribute with the specified value at the end.

`$("[attribute^='value']")` - selects a set of elements that have the specified attribute with the specified value at the start.

## Demo: change style using jQuery



```
<!DOCTYPE html>
<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js">
  </script>
</head>
<body>
  <h1>Welcome to My Homepage</h1>
  <p id="intro">I have a feeling</p>
  <p>The feeling is good</p>
  <p>Try all the different attributes</p>
  <p>Who is your favourite:</p>
  <ul id="choose">
    <li>That</li>
    <li>Those</li>
    <li>Them</li>
  </ul>
  <script>
  $(document).ready(() => {
    $("[id]").css("background-color", "skyblue");
  });
</script>
</body>

</html>
```



Welcome to My Homepage

I have a feeling

The feeling is good

Try all the different attributes

Who is your favourite:

- That
- Those
- Them

## Parents vs. Children

- Select elements based on their **hierarchical relationship**.
- Referred to as jQuery child selectors or parent selectors.

`$("parent>child")` - combines two selectors: a jQuery child selector and a parent selector. They select the elements specified as *child* - ones that are children of elements specified as *parent*.

`$("ancestor descendant")` - combines two selectors to select all elements specified as *descendant* - ones that are below the elements specified as *ancestor* in the node relationships.

`$(":root")` - selects the document's root element.

`$(":parent")` - selects elements that have at least a single child node.

`$(":empty")` - selects all elements that have no children (this would include text nodes as well).

# The Keyword-Based Types

- Some jQuery selectors are keyword-based. You may recognize them easily, as they are always preceded by a colon (:).

  `$(":button")` - selects button elements and elements that have the type attribute with the value button.

  `$(":radio")` - selects radio elements.

  `$(":checkbox")` - selects all checkbox elements.

  `$(":checked")` - selects all selected or checked elements.

  `$(":disabled")` - selects all disabled elements.

  `$(":file")` - selects all element of the *file* type.

  `$(":submit")` - selects elements of the *submit* type.

  `$(":header")` - selects header elements (h1, h2, h3, h4, h5, h6).

  `(":image")` - selects all image elements.

  `$(":input")` - selects all input, select, textarea and button elements.

  `$(":text")` - selects all elements of *text* type.

  `$(":reset")` - selects all elements of *reset* type.

# Other jQuery Selectors

- Nth of *, only, first and last selectors
  - Ex: `$(":nth-child(n)")` - selects the *n*th children of the specified parent elements.

- Other selectors
  - Ex: `$("prev + next")` - selects the next element adjacent to the element specified as prev, which matches the type specified by selector next.

- Find out more at
  - http://api.jquery.com/category/selectors/

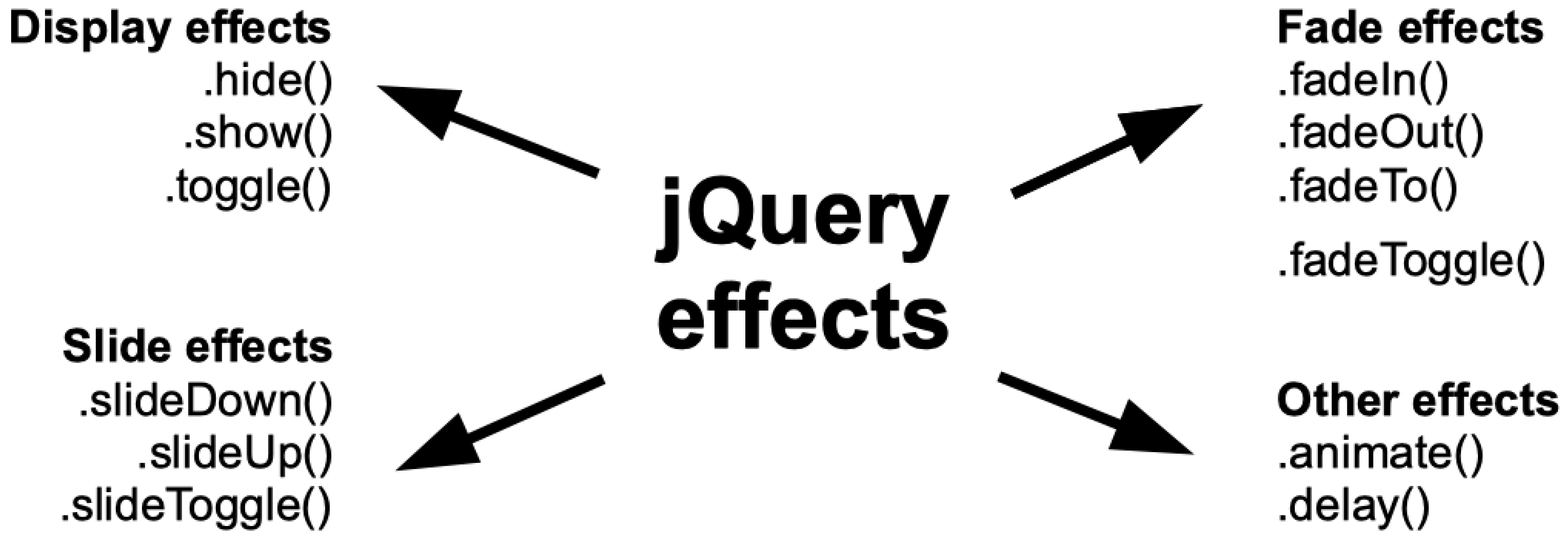
# jQuery / DOM comparison

| DOM method | jQuery equivalent |
|---|---|
| getElementById("id") | $("#id") |
| getElementsByTagName("tag") | $("tag") |
| getElementsByName("somename") | $("[name='somename']") |
| querySelector("selector") | $("selector") |
| querySelectorAll("selector") | $("selector") |

## jQuery Events

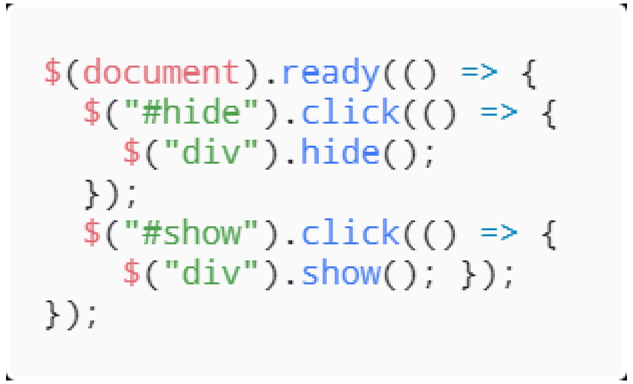
- In JavaScript and jQuery events can also be called user interactions.
- The term refers to an action of the user interacting with the browser.
- It is registered by an event listener, which can have functions assigned to specify how it reacts to the event.
- Simple examples of jQuery events include:
  - Moving the mouse over an element
  - Clicking an element
  - Pressing a key

# jQuery Effects

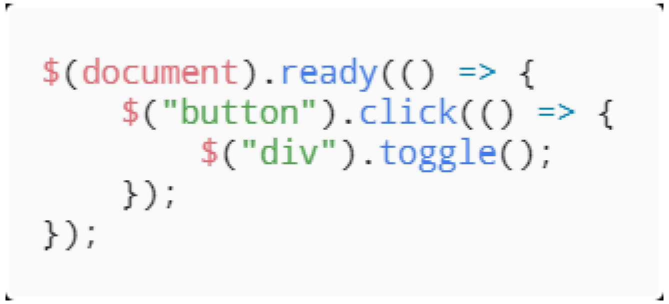


## jQuery Show & Hide

- jQuery has a selection of various methods for applying **effects** and **animation** to elements.
- The **hide** and **show** methods might be considered the most basic, as you can apply them with a minimal amount of code.
- jQuery hide show can also be combined with **toggle** method.



```
$(document).ready(() => {
  $("#hide").click(() => {
    $("div").hide();
  });
  $("#show").click(() => {
    $("div").show(); });
});
```



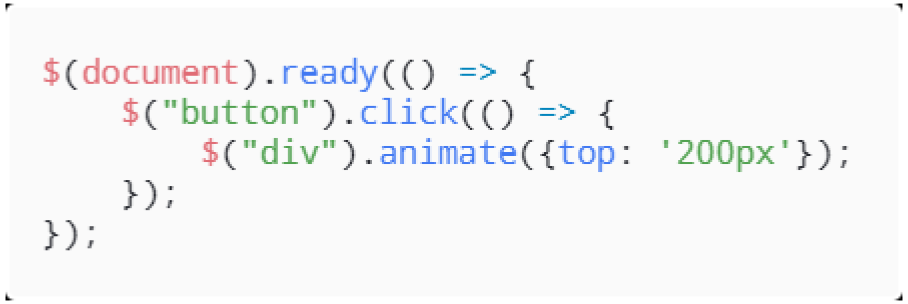
```
$(document).ready(() => {
    $("button").click(() => {
        $("div").toggle();
    });
});
```

**toggle**

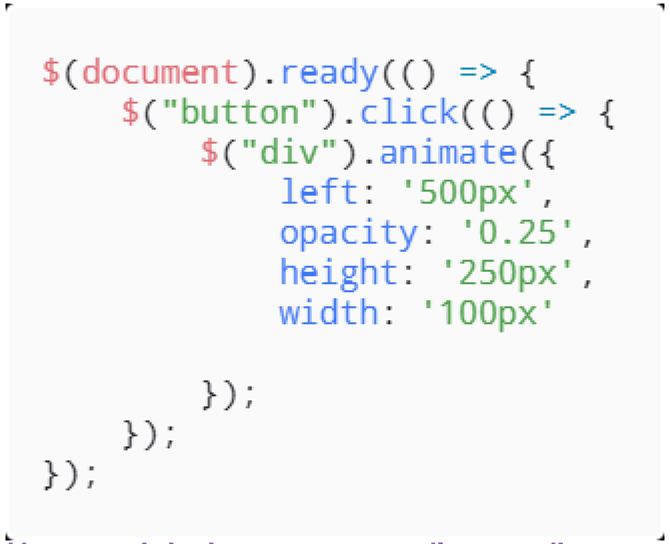
See the demo: https://www.bitdegree.org/learn/jquery-show-hide

## jQuery Animate

- The jQuery **animate** method is used to **animate the CSS values** of an object.
- Before you use animate in jQuery, you need to make sure particular values are animatable.



```
$(document).ready(() => {
    $("button").click(() => {
        $("div").animate({top: '200px'});
    });
});
```

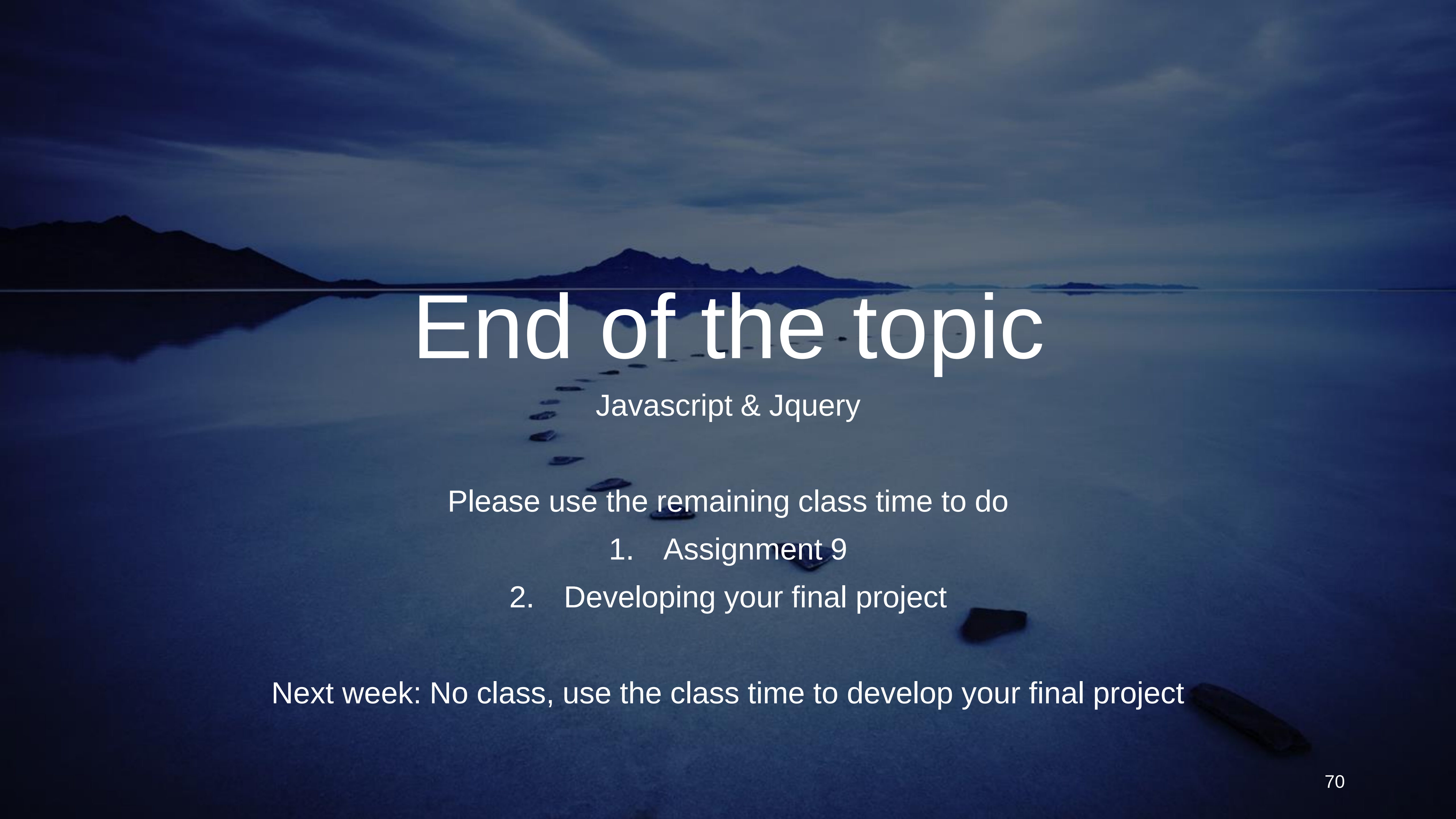


```
$(document).ready(() => {
    $("button").click(() => {
        $("div").animate({
            left: '500px',
            opacity: '0.25',
            height: '250px',
            width: '100px'

        });
    });
});
```

See the demo: https://www.bitdegree.org/learn/jquery-animate

## jQuery tutorials

- jQuery
    - https://learn.jquery.com/
- W3School
    - https://www.w3schools.com/jquery/default.asp



# End of the topic

Javascript & Jquery

Please use the remaining class time to do

1. Assignment 9
2. Developing your final project

Next week: No class, use the class time to develop your final project