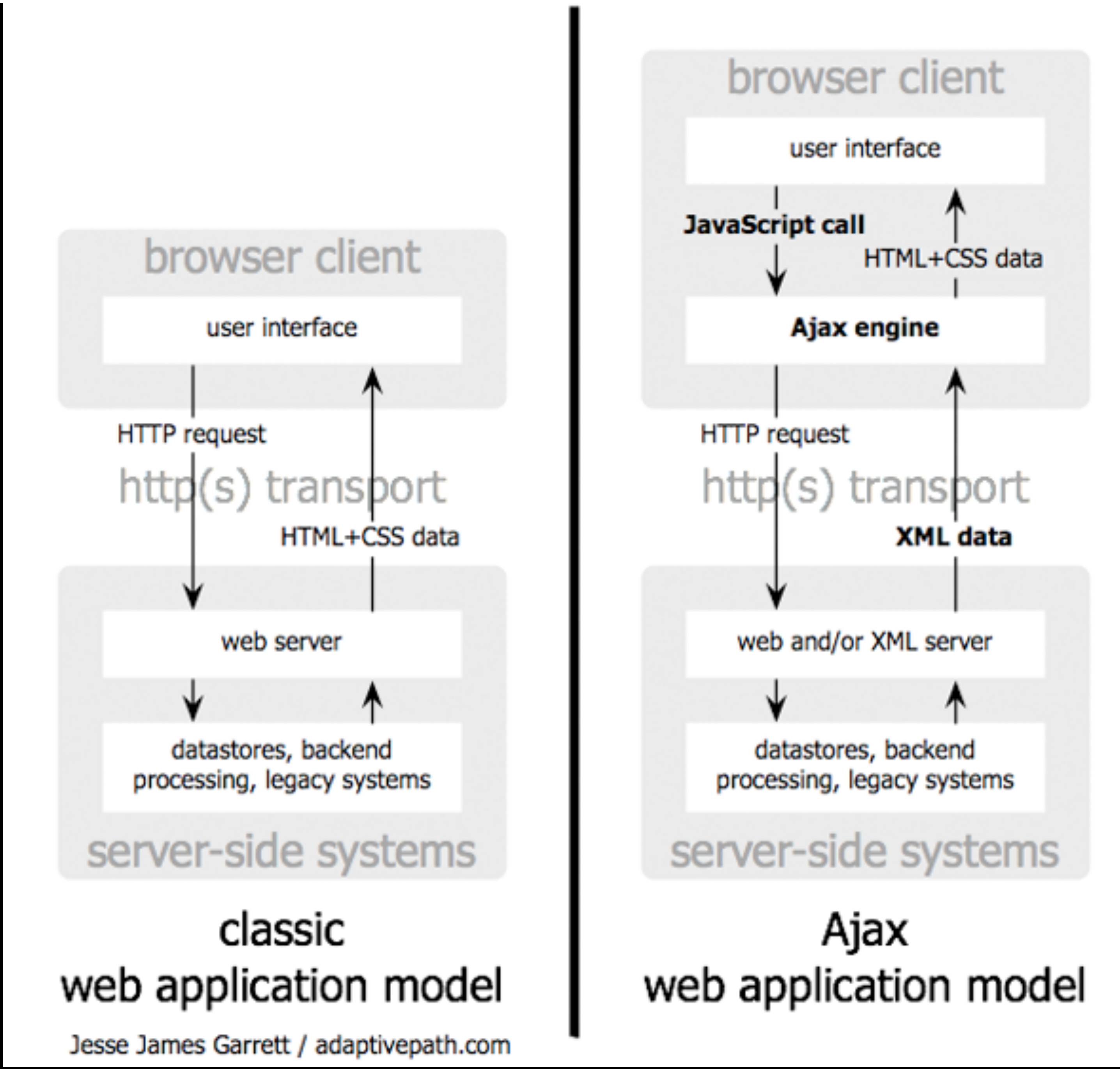
Intro 2 JS

The collection of technologies used by Google was christened **AJAX**, in a seminal essay by Jesse James Garrett of web design firm Adaptive Path.

"**Ajax** isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways.

Ajax incorporates:

- standards-based presentation using **XHTML** and **CSS**;
- dynamic display and interaction using the Document Object Model (DOM);
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- + **JavaScript** binding everything together."



Jesse James Garrett, 2005

(To view this link you'll need the Wayback Machine)

Server Side Programming

Unitl now, we have been learning about *client side* web development. The client side, in this case, is the web browser and the technologies that go into it (HTML, CSS, JavaScript).

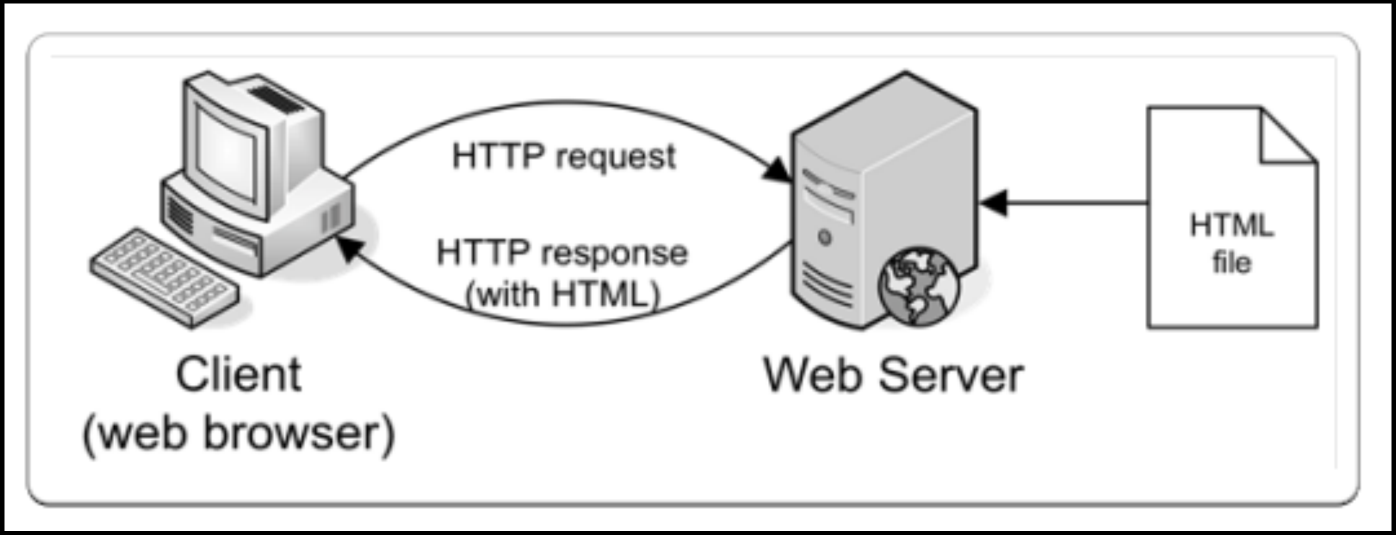
Static vs. Dynamic Web Pages

Server side programming languages (e.x. JavaScript, PHP, .NET, Python, Perl) make creating dynamic web pages possible. Using HTML and CSS we have been creating static web pages in this class. Static web pages will always contain the same content or information in them no matter what, until you change the source code of the page. Dynamic web pages are capable of producing different content for a web page or web site using the same source code, based on the application logic written by the developer. (RSS feeds were one of the first instances of this w/ **Web 2.0**).

Dynamic web pages will often be powered by a server side programming language and a database (e.x. JavaScript, MySQL, SQL Server, Oracle). Together, these technologies can determine what information should be returned to the browser and construct dynamic content for the client side to present to the user.

Request-Response Procedure

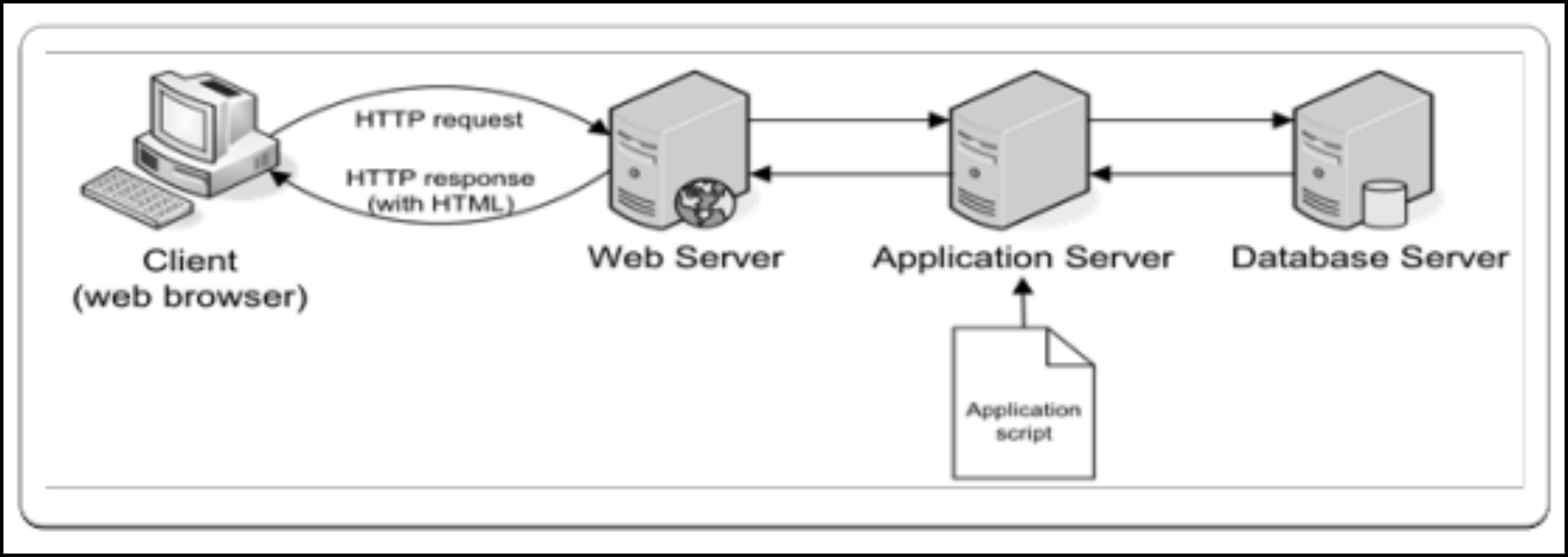
The HTTP request-response procedure explains how you are able to access web sites over the internet. When working with regular client side technologies, the request-response cycle is pretty simple.



1. You enter a website address http://example.com into the browser's address bar.
2. The browser looks up the address for http://example.com and requests the web page associated with it.
3. The request traverses the internet for the server that lives at http://example.com
4. The server at http://example.com receives the request and sends the (HTML) web page for http://example.com.
5. The browser receives the web page sent from the server and shows it to the user.

Server Side

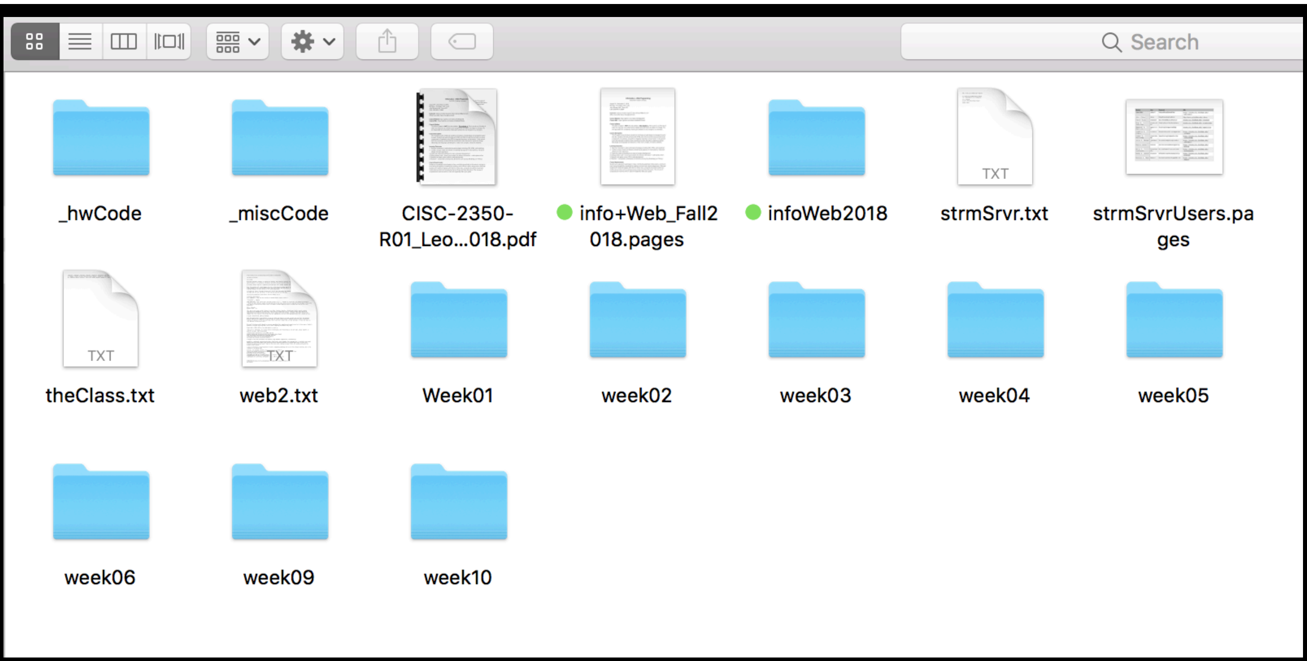
1. You enter a website address http://example.com into the browser's address bar.
2. The browser looks up the address for http://example.com and requests the web page associated with it.
3. The request traverses the internet for the server that lives at http://example.com.
4. The server at http://example.com receives the request and interprets it as a PHP file.
5. The PHP interpreter executes the PHP code and passes any MySQL statements to the database to be processed.
6. The PHP interpreter and MySQL return the resulting data to the web server as HTML.
7. The web server sends the generated page to the browser, which displays it to the user.



“Think like a computer”

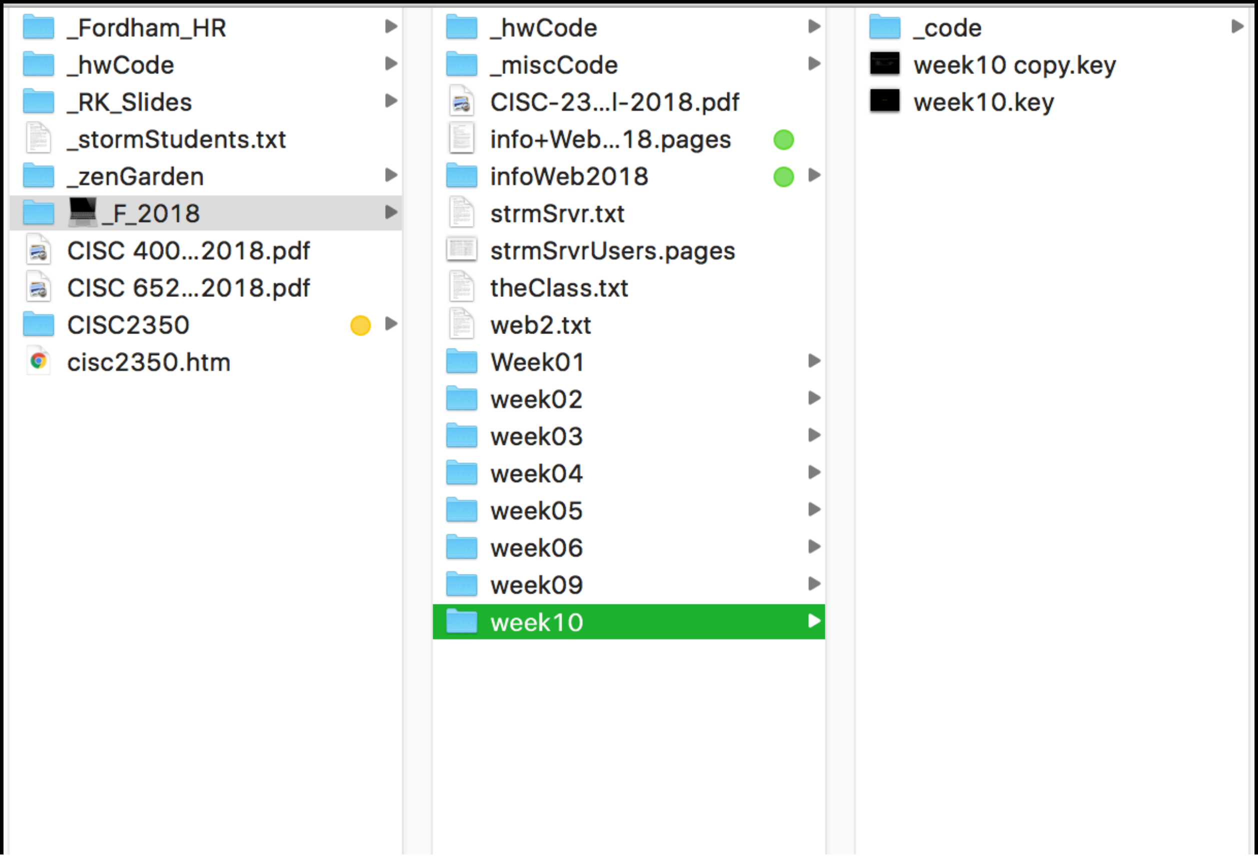
Computers solve problems programmatically; they follow a series of instructions, one after another. The type of instructions they need are often different from the type of instructions yo
might give to another human.

It's as simple as the difference btw this


Search

_hwCode	_miscCode	CISC-2350-R01_Leo...018.pdf	● info+Web_Fall2018.pages	● infoWeb2018	strmSrvr.txt	strmSrvrUsers.pages
theClass.txt	web2.txt	Week01	week02	week03	week04	week05
week06	week09	week10				

+ this


_Fordham_HR	_hwCode	_code
_hwCode	_miscCode	week10 copy.key
_RK_Slides	CISC-23...l-2018.pdf	week10.key
_stormStudents.txt	info+Web...18.pages ●	
_zenGarden	infoWeb2018 ●	
_F_2018	strmSrvr.txt	
CISC 400...2018.pdf	strmSrvrUsers.pages	
CISC 652...2018.pdf	theClass.txt	
CISC2350 ●	web2.txt	
cisc2350.htm	Week01	
	week02	
	week03	
	week04	
	week05	
	week06	
	week09	
	week10	

Computers create models of the world using data.

Object-oriented programming

Objects (Things)

In computer programming, each physical thing in the world can be presented as an object. Each object can have its own:

Properties
Events
Methods

Properties

Each property has a **name** and a **value** + each of these name/value pairs tells you something about each individual instance of that object

Events

Programs are designed to do different thing when users interact with the computer in different ways. For example, clicking on a link could bring up another webpage.

An event is a the computer's way of sticking up its hadn't to say, "hey that just happened."

Programmers choose which events they respond to. When a specific event happens, that event can be use to trigger a speck section of the code.

Methods

Methods represent things people need to do with object. They can retrieve or update the values of an object’s properties.

They are questions + instructions that:

- Tell you something about the object (using info stored in its properties)
- Change the value of one or more of the object’s properties

The code for a method can contain lots of instructions that together represent one task (ex: changeSpeed()).

When you use a method, you do not always need to know **how** it achieves its task; you just need to know **how to ask the question** and **how to interpret any answers it gives you**.

The events, methods + properties of an object all relate to each other. Events can trigger methods, and methods can retrieve or update an objects' properties.

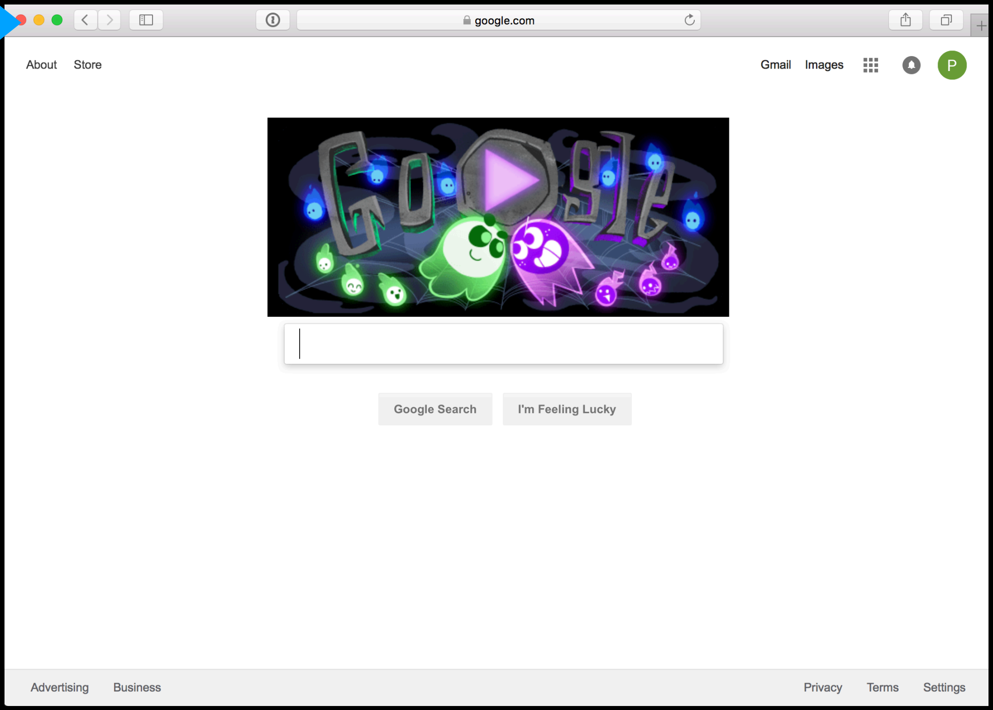
1. Event: button clicked;
2. Calls method changeBackgroundColor();
3. Updates property backgroundColor to blue;

Window Object

The browser represents each window or tab using a **window object**. The **location property** of the **window object** will tell you the URL of the current page.

Object type: window
Properties:

location	http://www.google.com



try this command in the window inspect console: **console.log(window.location)**
(it will return the current url)

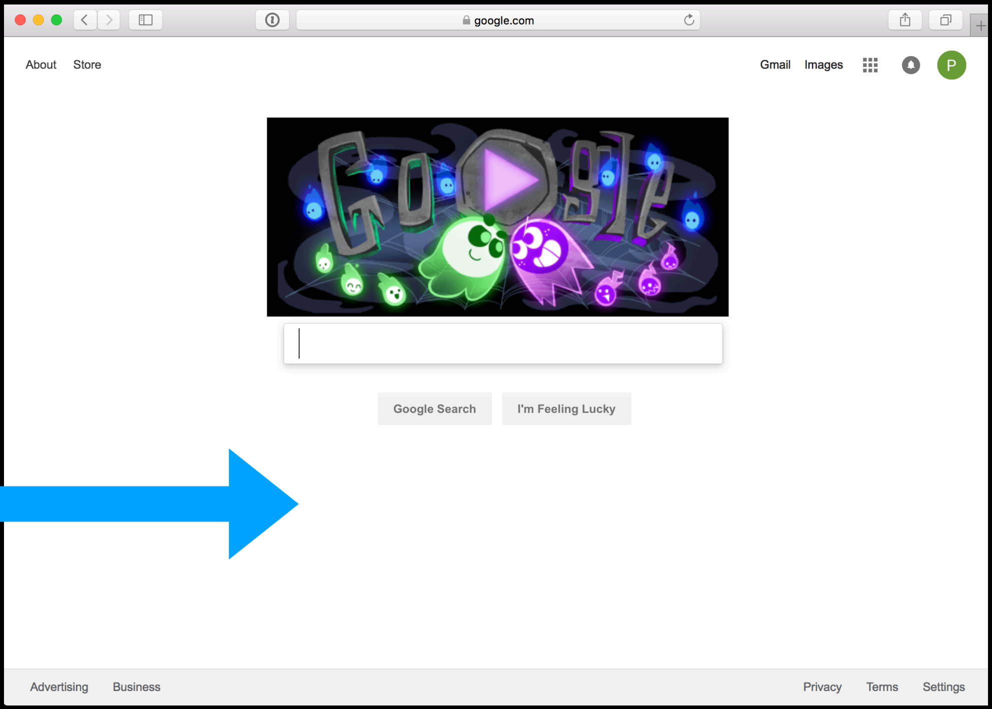
Document Object

The current webpage loaded into each window is modeled using a document object.

Object type: document

Properties:

URL	http://www.google.com
lastModified	10/31/18 6:12 pm
title	Google Search Home Page



try this command in the window inspect console: **console.log(document.title)**
(it will return the current url)

Scripts

A script is a set of instructions that a computer can follow one-by-one. Each individual instruction or step is known as a STATEMENT.

Scripts

You need to start with the big picture of what you want to achieve, and break that down into smaller steps:

1. Define the goal
2. Design the script
3. Code each step

What JavaScript Can Do:

Access content: you can use JavaScript to select any element, attribute or text from an HTML page.

Modify content: you can use JavaScript to add elements, attributes, and text to the page or remove them.

Program rules: you can specify a set of steps for the browser to follow, which allows it to access or change the content of a page.

React to events: you can specify that a script should run when a specific event has occurred.

JS Syntax

1. A semicolon notes that a step is over + JavaScript interpreter should move to the next step
2. Each statement starts on a new line
3. JavaScript is case sensitive: myName is not equal to myname or MYNAME
4. Some statements can be organized into code blocks:

```
if (value > theVariable) {
    //do this;
}
```

notice no ; after the code block

JS Syntax

1. You should ALWAYS write comments to explain what your code does (especially JS logic)
2. There can be single-line and multi-line comments
3. Single-line comments are written with two forward
   slashes //

   Often used for short descriptions of what the code is doing
4. Multi-line comments are written using /* and */ characters

   Often used for descriptions of how the script works, or to prevent a section of the script from running when testing it

JS Syntax

1. Once you created a variable, you need to tell what information it should store (assign a value)

2.To announce it you need to create a variable and give it a name (declare it)
Until you have assigned a value, it’s undefined

```
theSpeed = 7;
```

theSpeed is the variable name
= is the assignment operator
7 is the variable value

JS Syntax

Variables

Variables are containers that you can store values in. You start by declaring a variable with the var keyword, followed by any name you want to call it

```
var theVariable;
var theVariable = ‘Bob';                          ——> String Data
var theVariable = 10;                             ——> Numeric Data
var theVariable =  true;                          ——> Boolean
var theVariable = [536, 3, 3354684, 325]  ——> Array
```

JS Syntax

let

allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. This is unlike the **var** keyword, which defines a variable globally, or locally to an entire function regardless of block scope.

At the top level of programs and functions, let, unlike var, does not create a property on the global object.

** Note: the browser is not able to identify local variables.

JS Syntax

const

Constants are block-scoped, much like variables defined using the let statement. The value of a constant cannot change through reassignment, and it can't be redeclared.

Data Type: Numeric

1. Numeric data type handles numbers
2. Use **typeof** to log what kind of data type you're using

```
var thePrice;
var theQuantity;
var theTotal;

thePrice = 5;
theQuantity= 14;
theTotal = thePrice * theQuantity;

console.log(typeof theTotal);
```

Data Type: Strings

1. The string data type consists of letters and other characters
2. They have to be enclosed with a pair of quotes (single or double)
   Opening quote must match the closing quote
3. They can be used working with any kind of text

```
var myName;
var myMessage;

myName = “Rebecca”;
myMessage = “Atlanta is a really great show y’all should check it out!”;

console.log(typeof myName);
```

Data Type: Boolean

1. Boolean data type can have one of two values: true or false
2. They are helpful when determining which part of a script should run (when code can tak
more than one path);

```
var todayIsCloudy;
todayIsCloudy = true;

if (todayIsCloudy == true) {
        // if true, show this
        console.log("Yes, today is cloudy!”);
        // otherwise show this
 } else {
        console.log("No, it's actually sunny today!");
 }
```

Shorthand for creating variables

You can use these shorthands to create variables:

1:
```
var speed = 5;
var quantity = 14;
var total = speed * quantity;
```

2:
```
var speed, quantity, total;
speed = 5;
quantity = 14;
total = speed * quantity;
```

3:
```
var speed = 5, quantity = 14;
var total = speed * quantity;
```

Changing the value of a variable

Once you have assigned a value to a variable, you can then change it later in a script

```
var speed = 5;
var quantity = 14;
var total = speed * quantity;

//maybe something changed here and the value has to change now

speed = 10;

var textToShow = document.getElementById("hello");
textToShow.innerHTML = "Total cost is: " + "$" + total;
```

Rules for naming variables

1. The name must begin with a letter, $ sign or an underscore _, it cannot start with a number (e.g. name, $name, _name) - I would avoid $, because it might confuse it with jquery

2. You cannot use dash - or a dot . in a variable name (e.g. my-name, my.name)

3. You cannot use keywords or reserved words as variable names (e.g. function, type, this, etc.) - it usually changes the color when you use it

4. All variables are case sensitive (it is bad practice to create the same name using different cases (e.g. myName & Myname) - do not start with a capital letter in general

5. Use a name that describes the kind of information that the variable stores (e.g. firstName, lastName)

6. If a variable name is made up of more than one word use capital letter for the first letter of every word after the first one or underscore (e.g. myFirstName, myLastName, my_first_name, my_last_name)

In your JS file, let’s try these commands:

```
console.log(“Hello World”);
console.logwindow.location);
console.log(document);
```

Operators allow us to create single values from one or more values.

Types of operators

Assignment operators (assign values to variables):

```
var theMovie = "Blade Runner";
```

Arithmetic operators (perform basic math):

```
var height = 50 * 3;
```

String operators (combine two strings):

```
var sentence = "My name is " + "Rebecca";
```

Comparison operators (compare two values and return true or false):

```
height = 50 > 3 (will return false)
```

Logical operators (combine expressions and return true or false:

```
height = (50 < 3) && (3 > 2) (will return true)
```

Arithmetic Operators in JS

NAME	OPERATOR	PURPOSE & NOTES	EXAMPLE	RESULT
ADDITION	+	Adds one value to another	10+5	15
SUBTRACTION	-	Substracts one value from another	10-5	5
DIVISION	/	Divides two values	10/5	2
MULTIPLICATION	*	Multiplies two values	10*5	50
INCREMENT	++	Adds one to the current number	i=10; i++;	11
DECREMENT	- -	Subtracts one from the current number	i=10; i- -;	9
MODULUS	%	Divides two values and returns the remainder	10%3;	1

String Operators in JS

There is only one string operator **+**

It’s used to join the strings together

```
var firstName = “Rebecca”;
var lastName = “Leopold”;

var fullName = firstName + lastName;
```

Process of joining two or more strings together into a new one is: **concatenation**.

If you’ll try to add other arithmetic operators on a string, it will return NaN

```
var fullName = firstName * lastName;
returns: “Not a Number”
```

Logical Operators

LOGIC	OPERATOR	EXAMPLE	NOTES
AND	&&	exprss1 && express2	Returns expr1 if it can be converted to **false**; otherwise, returns expr2. Thus, when used with Boolean values, **&&** returns **true** if both operands are **true**; otherwise, returns **false**.
OR	||	exprss1 || express2	Returns expr1 if it can be converted to **true**; otherwise, returns expr2. Thus, when used with Boolean values, || returns **true** if either operand is **true**.
NOT	!	! express	Returns **false** if its single operand can be converted to **true**; otherwise, returns **true**.

An expression results in a single value (produces a value and is written whenever a value is expected).

Types of expressions

Expressions that assign a value to a variable

```
var theMovie = “Blade Runner”;
```

Expressions that use two or more values to return a single value

```
var theHeight = 50 * 3;
var theSentence = “My name is ” + “Rebecca”;
```

Arrays

An array is a special type of variable. It doesn’t just store one value; it stores a list of values.

You should use an array whenever you’re working with a list of values that are related to each other. (ex: an array of images you want the user to click through, an array of colors to randomize a design)

Create an array and give it a name just like any other variable:

```
var theMovies = [ ];
```

2. Create an array using array literal technique:

```
var theMovies = [“Blade Runner”, “Sorry to Bother You”, “Groundhog Day”];
```

3. Create an array using array constructor technique:

```
var theMovies = new Array (“Blade Runner”, “Sorry to Bother You”, “Groundhog Day”);
```

Note: values in an array do **not have to be the same data type** (could be string, number, boolean in one array).

Note2: array literal is the preferred way to create an array in JS.

Values in Arrays

Values in an array are accessed through their numbers they are assigned in the list. The number is called index and starts from 0.

```
var theMovies = [“Blade Runner”, “Sorry to Bother You”, “Groundhog Day”];
                  index [0]         index [1]                index [2]
```

You can check the number of items in an array using **length** property

```
var theMoviesLength = theMovies.length;
```

Changing Values in Arrays

Let’s say we want to update the value of the second item (change “Sorry to Bother You” to something else)

To access the current third value, we have to call the name of the array followed by the index number for that value inside the square brackets:

```
var theMovies = [“Blade Runner”, “Sorry to Bother You”, “Groundhog Day”];
theMovies[1]
```

After we select a value, we can assign a new value to it:

```
theMovies[1] = “Clueless”;
```

When we log the updated array, we see updated values:

```
console.log(theMovies) ;
    [“Blade Runner”, “Clueless”,“Groundhog Day”];
```

Adding and removing values from the array

You can add values to the array using **.push()** method:

```
var theMovies = [“Blade Runner”, “Clueless”,“Groundhog Day”];
theMovies.push(“The Shining”);
```

You can remove values from the array using **.splice()** method:

```
var theMovies =  [“Blade Runner”, “Clueless”,“Groundhog Day”, “The Shining”];
theMovies.splice(0,1); (will remove “Blade Runner” movie)
```

Here - 0 is an index at what position an item should be removed and 1 how many items should be removed (in this case only one movie)

For Loops

A For loop uses a counter as a condition.
It instructs code to run a specified number of times.
Good to use when the number of repetitions is known, or can be supplied by the user.

```
var theYear = [“January”, “February”,   “March”, “April”, “May”, “June”, “July”, “August”, “September”,
“October”, “November”, “December”];
```

Keyword **Condition (counter)**

```
for (var i = 0; i < theYear.length; i ++){
        console.log(theYear[i]);
    }
```

code to execute during loop

For Loops

We can use for loops to programmatically work through arrays + get their values.

```
var theYear = [“January”, “February”,   “March”, “April”, “May”, “June”, “July”, “August”, “September”,
“October”, “November”, “December”];
```

Condition (counter)

Keyword

```
for (var theIndex = 0; theIndex < theYear.length; theIndex ++){
          console.log(theYear[theIndex]);
       }
```

code to execute during loop

While Loops

Good to use in applications with numeric situations and when we don’t know how many times the code should run.
In other words: the loop repeats until a certain “condition” is met.
If the condition is false at the beginning of the loop, the loop is never executed.

```
var theIndex = 1;

while ( theIndex < 10 ){
  console.log( theIndex );
  theIndex ++;
}
```

Do while loop

Same concept as the while loop.
Except that this loop will always execute the loop at least once (even if the condition is false).
Good to use when you are asking a question, whose answer will determine if the loop is repeated.

```
var i = 1;

do {
  console.log( i );
  i ++;
} while (i < 10);
```

Functions group a series of statements together to perform a specific task.

Prgrmmng Vocabulary

When you ask function to perform its task, you **call** a function

Some functions need to be provided with information in order to achieve a given task - pieces of information passed to a function are called **parameters**

When you write a function and expect it to provide you with an answer, the response is known as **return value**

Declaring a function

To create a function you give it a name and write statements inside to achieve a task you want it to achieve

function keyword **function name**

code statement

```
function buttonClicked() {

    console.log(“hello”);


}
```

code block inside curly braces

Calling a function

You call a function by writing it’s name somewhere in the code.

```
function buttonClicked() {
    console.log(“hello”);
}


buttonClicked() // code after
```

Declaring functions with parameters

Sometimes a function needs specific information to perform it’s task **(parameters)**
Inside the function the parameters act as variables

parameters

```
function countTotal(itemNumber, price) {
    return itemNumber * price;
}
```

parameters are used like variables inside the function

Calling functions with parameters

When you call a function that has **parameters,** you need to specify the values it should take in. Those values are called **arguments.**

Arguments are written inside parentheses when you call a function + can be provided as values or as variables

```
function countTotal(itemNumber, price) {
        return itemNumber * price;
}

countTotal(7,15); //will return 105

(itemNumber = 7 * price = 15) countTotal(itemNumber, price);
```

Using “return” in a function

return is used to return a value to the code that called the function

The interpreter leaves the function when return is used and goes back to the statement that called it

```
function countTotal(itemNumber, price) {
  return itemNumber * price;
  // interpreter will skip any code found after return
 }
```

Variable Scope

Where you declare a variable affects where it can be used within your code

If it’s declared inside a function, it can only be used inside that function

It’s known as variable’s **scope**

Local + Global Variables

When a variable is created inside a function
It’s called **local variable** or **function-level variable**

When a variable is created outside a function
It’s called **global variable** can be called anywhere
the in code + will take up more memory

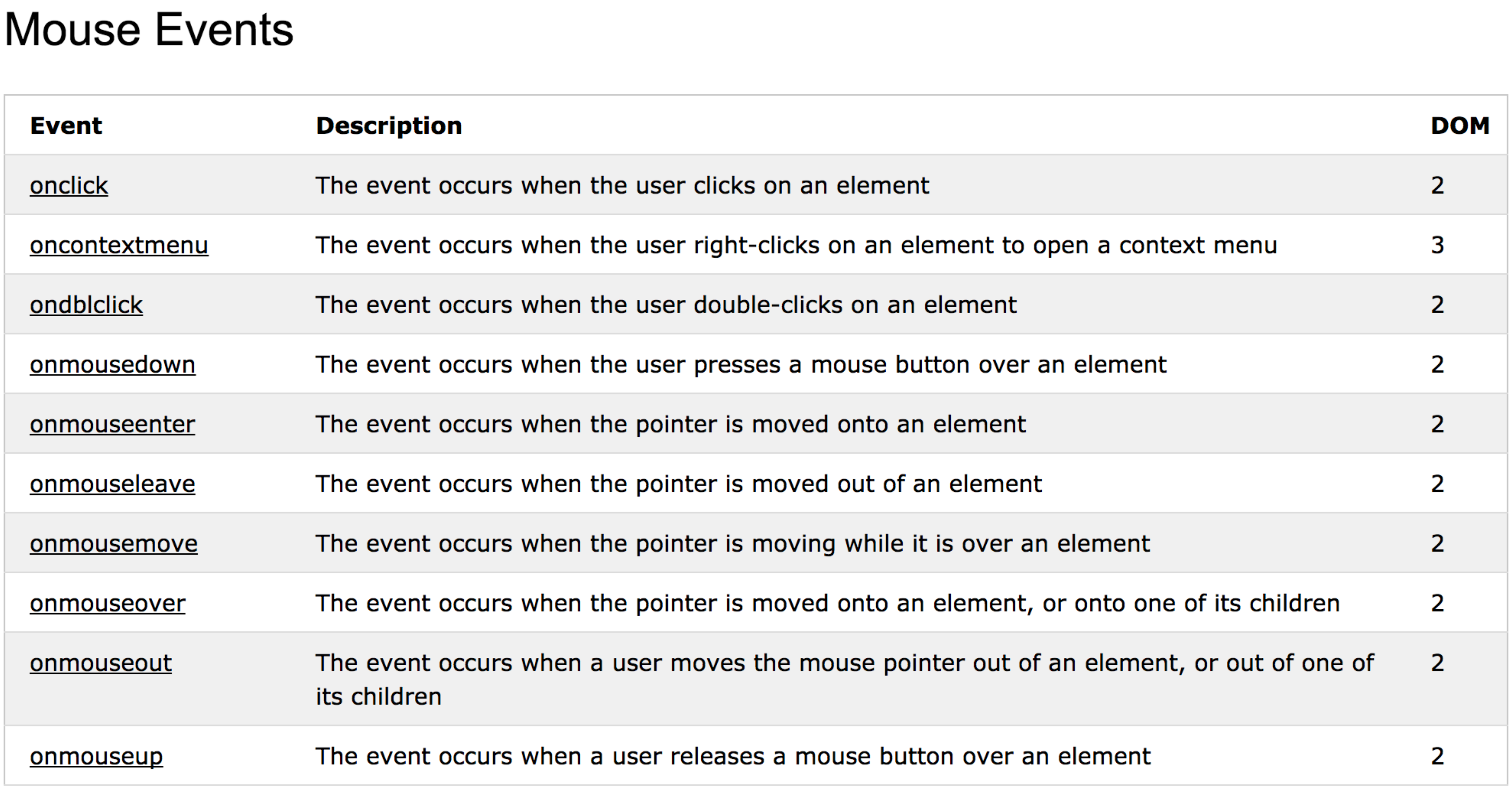
```
var salesTax = .08

function countTotal(itemNumber, price) {
	var theSum = itemNumber * price;
	var theTax = theSum * salesTax;
	var theTotal = theSum + theTax;
	return theTotal;
}

console.log(typeof salesTax);
```

When an HTML document is loaded into a web browser, it becomes a **document object**. The document object provides properties and methods to access all node objects, from within JavaScript.

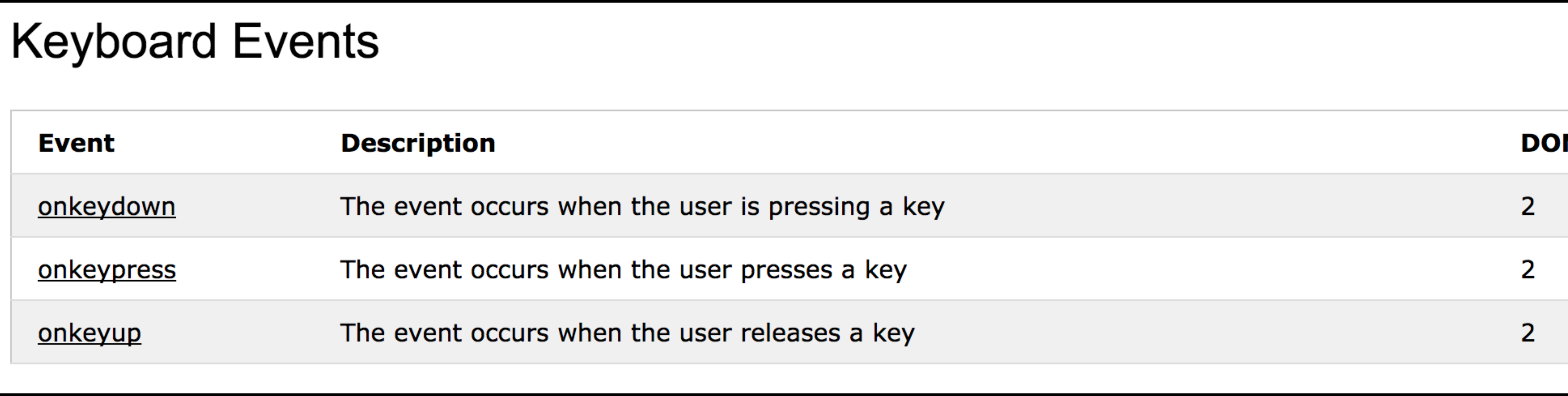
Built-in DOM events



Mouse Events

Event	Description	DOM
onclick	The event occurs when the user clicks on an element	2
oncontextmenu	The event occurs when the user right-clicks on an element to open a context menu	3
ondblclick	The event occurs when the user double-clicks on an element	2
onmousedown	The event occurs when the user presses a mouse button over an element	2
onmouseenter	The event occurs when the pointer is moved onto an element	2
onmouseleave	The event occurs when the pointer is moved out of an element	2
onmousemove	The event occurs when the pointer is moving while it is over an element	2
onmouseover	The event occurs when the pointer is moved onto an element, or onto one of its children	2
onmouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children	2
onmouseup	The event occurs when a user releases a mouse button over an element	2

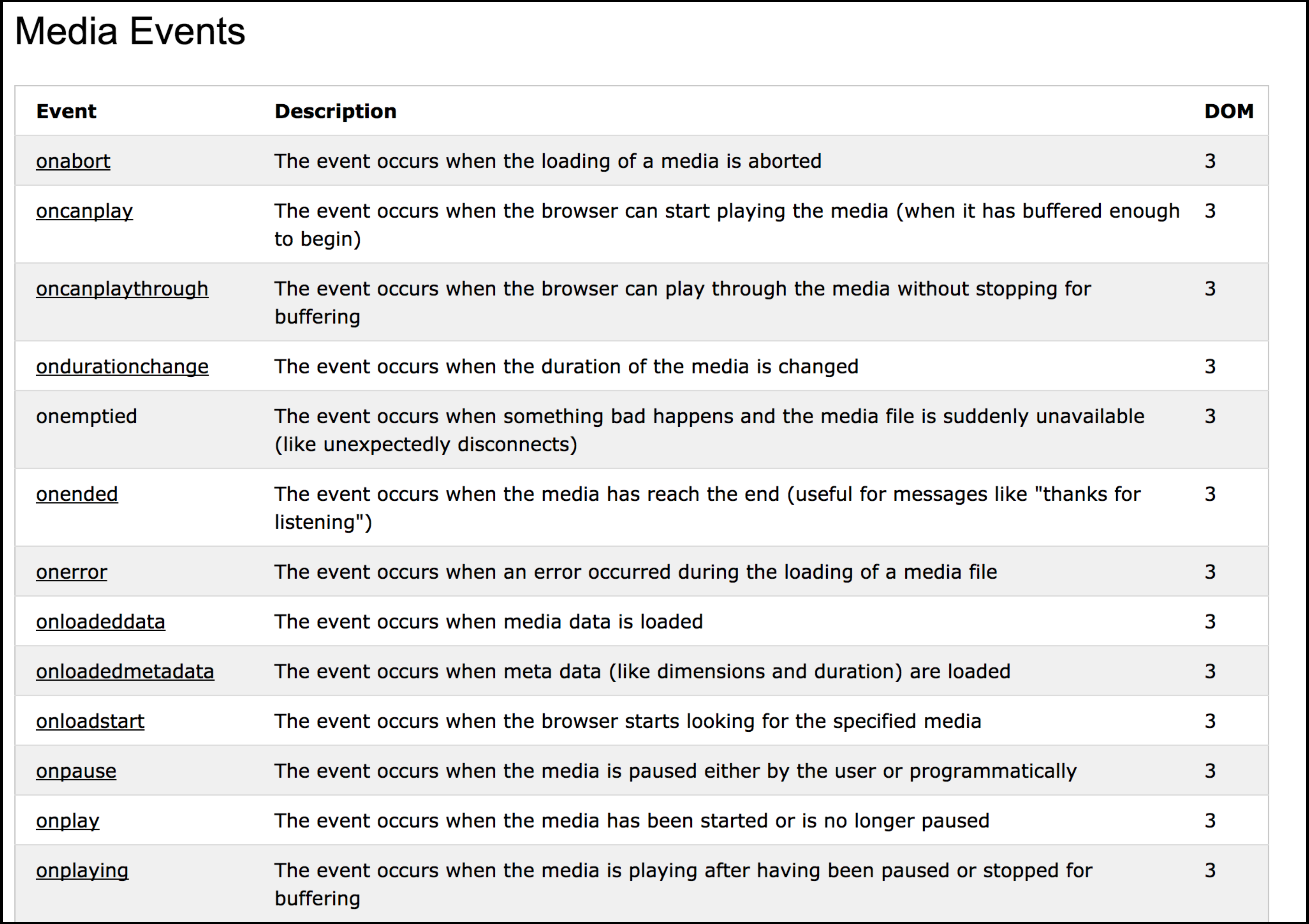
Built-in DOM events



Keyboard Events

Event	Description	DOI
onkeydown	The event occurs when the user is pressing a key	2
onkeypress	The event occurs when the user presses a key	2
onkeyup	The event occurs when the user releases a key	2

Built-in DOM events



Media Events

Event	Description	DOM
onabort	The event occurs when the loading of a media is aborted	3
oncanplay	The event occurs when the browser can start playing the media (when it has buffered enough to begin)	3
oncanplaythrough	The event occurs when the browser can play through the media without stopping for buffering	3
ondurationchange	The event occurs when the duration of the media is changed	3
onemptied	The event occurs when something bad happens and the media file is suddenly unavailable (like unexpectedly disconnects)	3
onended	The event occurs when the media has reach the end (useful for messages like "thanks for listening")	3
onerror	The event occurs when an error occurred during the loading of a media file	3
onloadeddata	The event occurs when media data is loaded	3
onloadedmetadata	The event occurs when meta data (like dimensions and duration) are loaded	3
onloadstart	The event occurs when the browser starts looking for the specified media	3
onpause	The event occurs when the media is paused either by the user or programmatically	3
onplay	The event occurs when the media has been started or is no longer paused	3
onplaying	The event occurs when the media is playing after having been paused or stopped for buffering	3

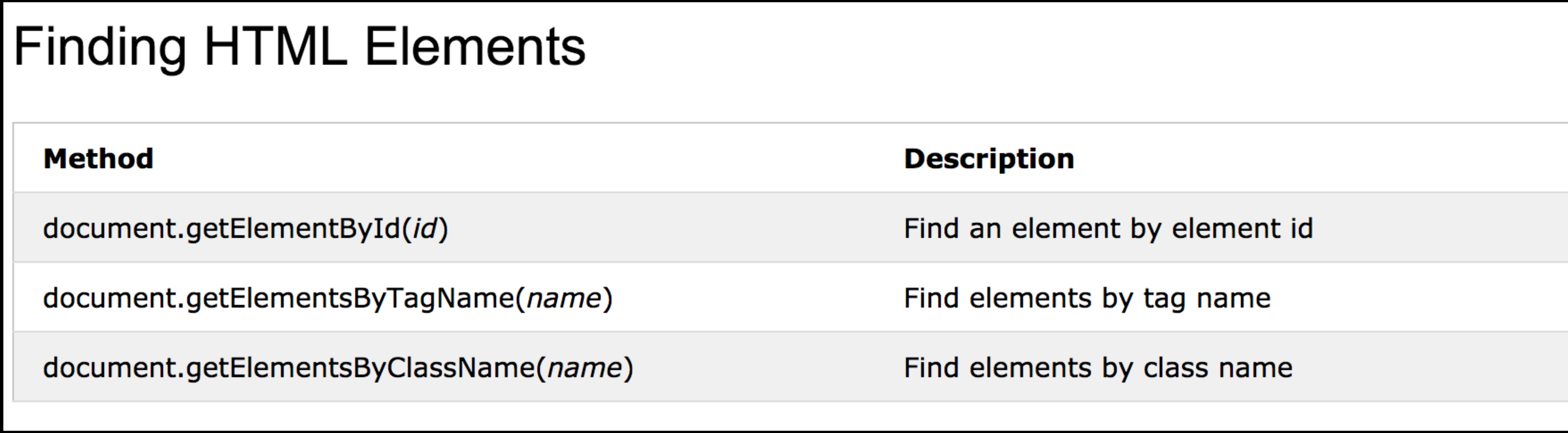
Built-in DOM properties + methods

HTML DOM **methods** are actions you can perform (on HTML Elements).
HTML DOM **properties** are values (of HTML Elements) that you can set or change.

```
document.getElementById(“hello”).innerHTML = “Hello!”;
```

getElementById -> method
innerHTML -> property

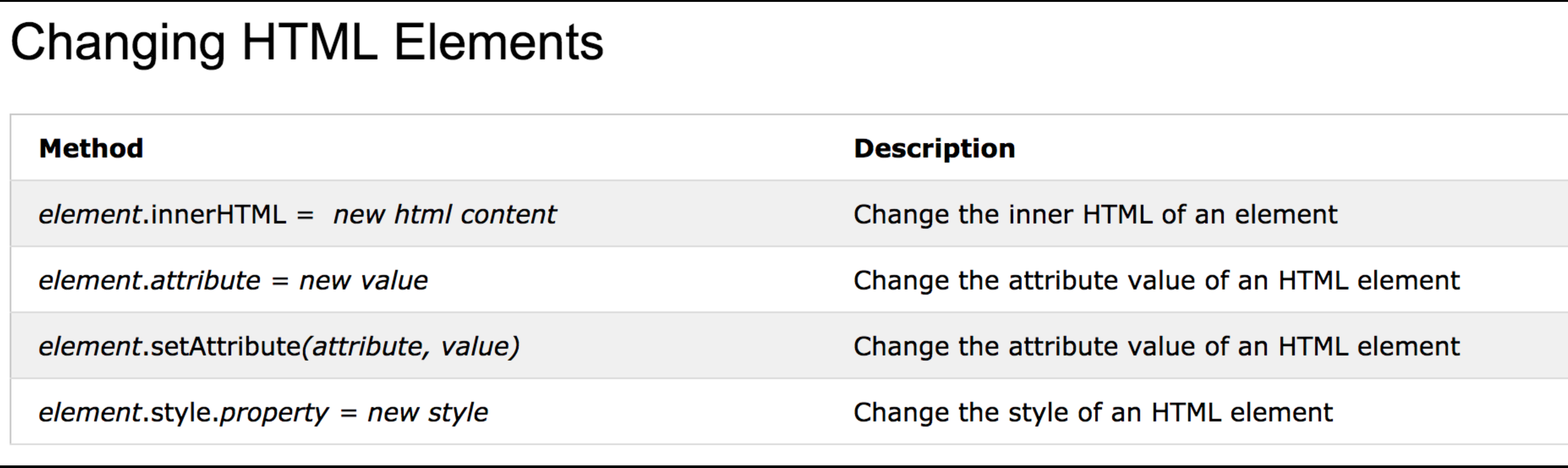
Built-in DOM properties + methods



Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

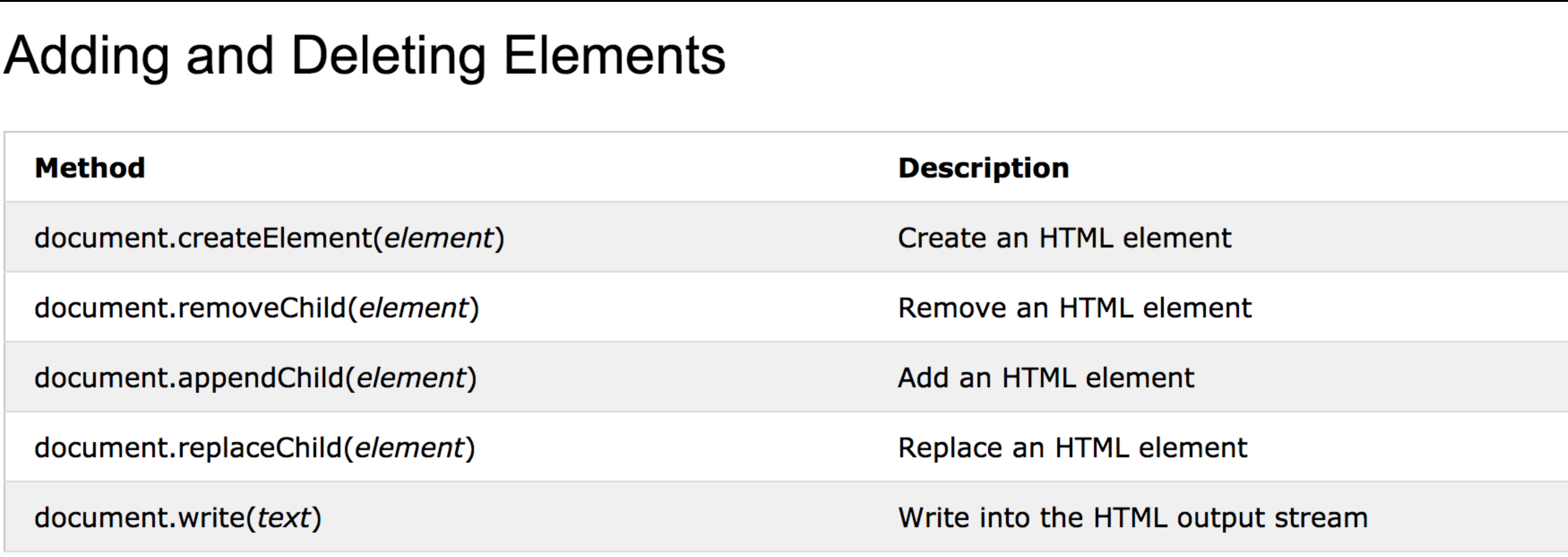
Built-in DOM properties + methods



Changing HTML Elements

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Built-in DOM properties + methods



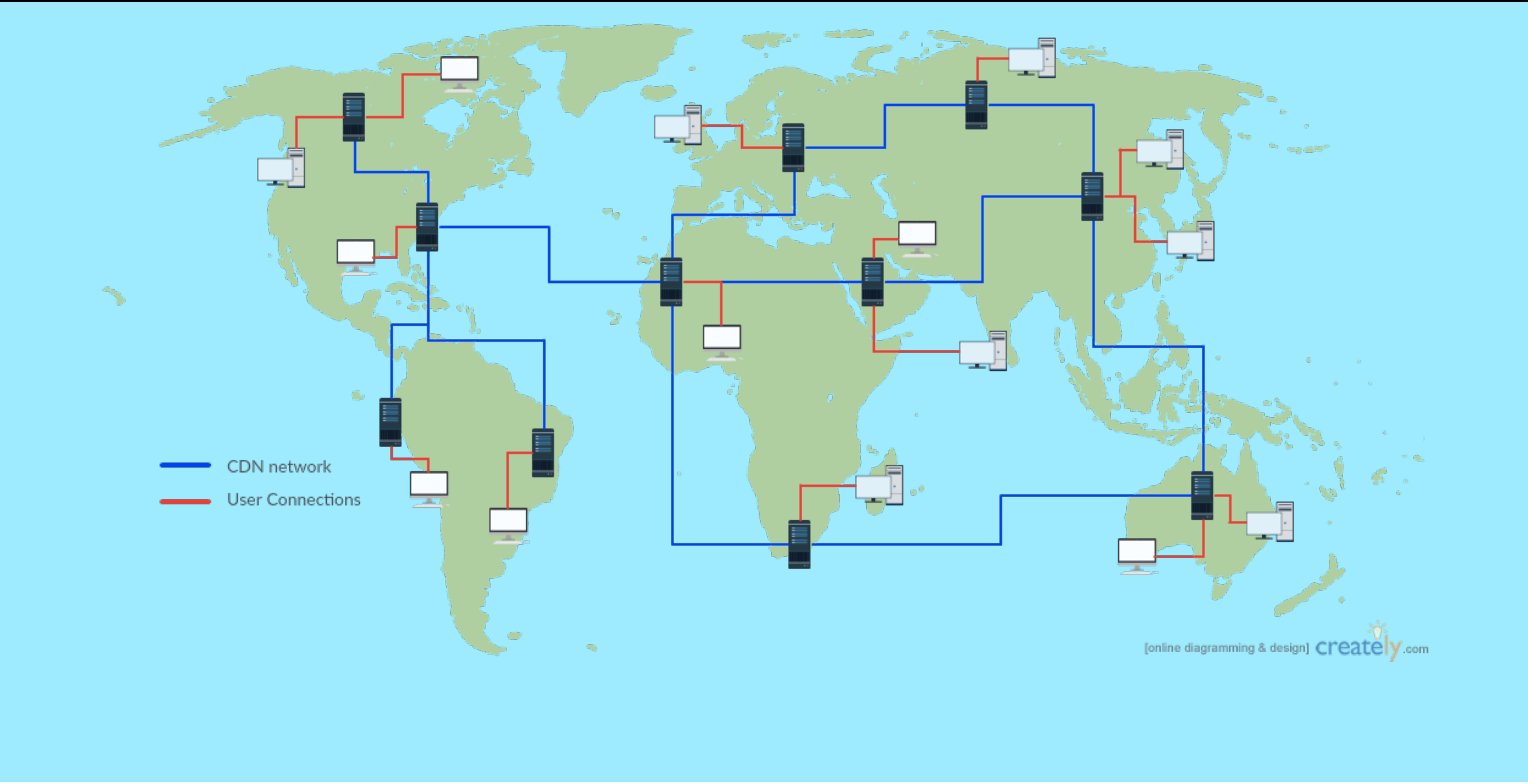
Adding and Deleting Elements

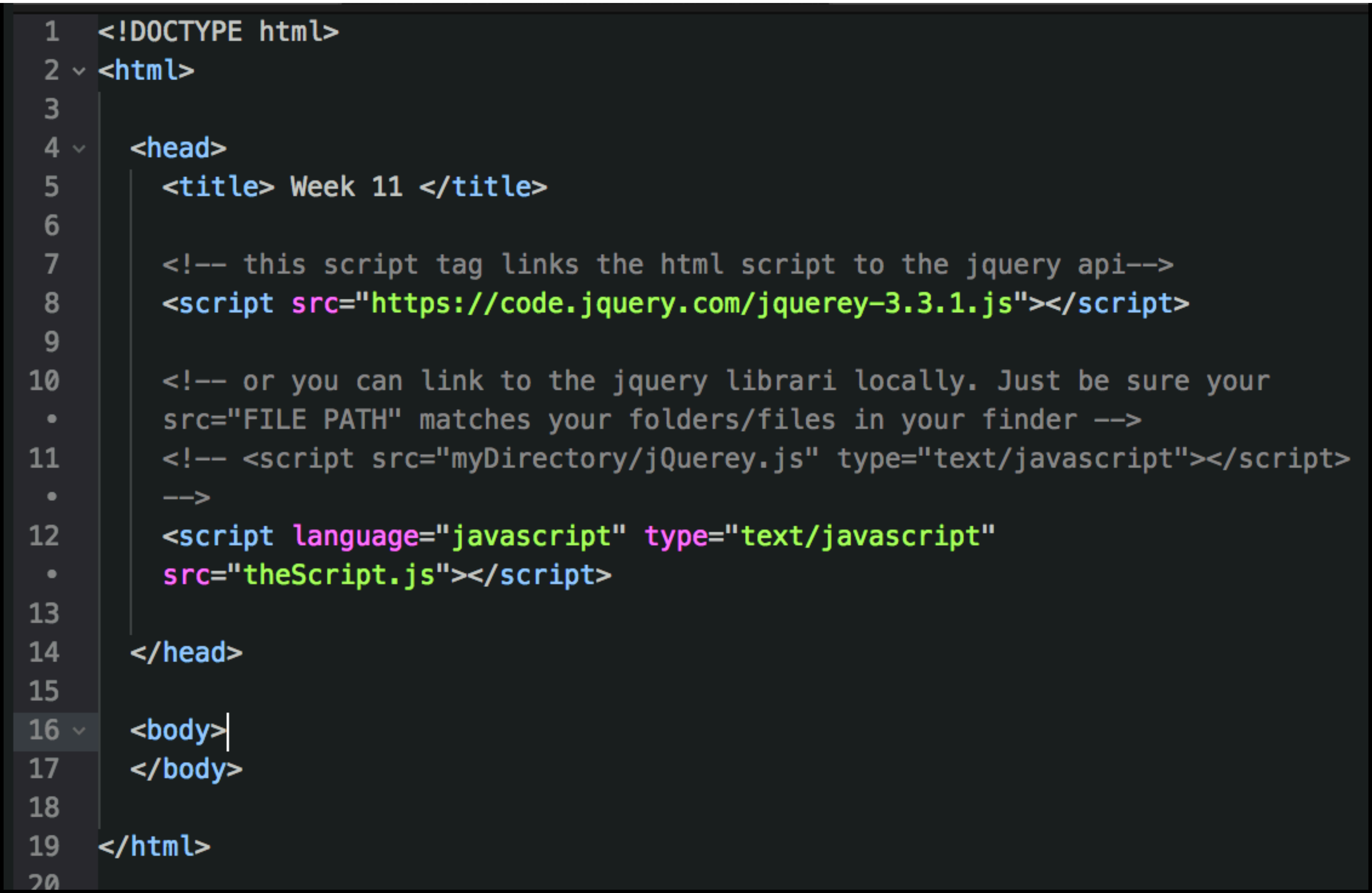
Method	**Description**
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Using Libraries

1. Download a local copy of the JS
2. Use a CDN (via the libraries API)

A 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 p5 or jQuery file via CDN, it will potentially arrive faster and more efficiently to the user than if you hosted it on your own server.





```
<!DOCTYPE html>
<html>

  <head>
    <title> Week 11 </title>

    <!-- this script tag links the html script to the jquery api-->
    <script src="https://code.jquery.com/jquerey-3.3.1.js"></script>

    <!-- or you can link to the jquery librari locally. Just be sure your
    src="FILE PATH" matches your folders/files in your finder -->
    <!-- <script src="myDirectory/jQuerey.js" type="text/javascript"></script>
    -->
    <script language="javascript" type="text/javascript"
    src="theScript.js"></script>

  </head>

  <body>
  </body>

</html>
```

dwnld jquery