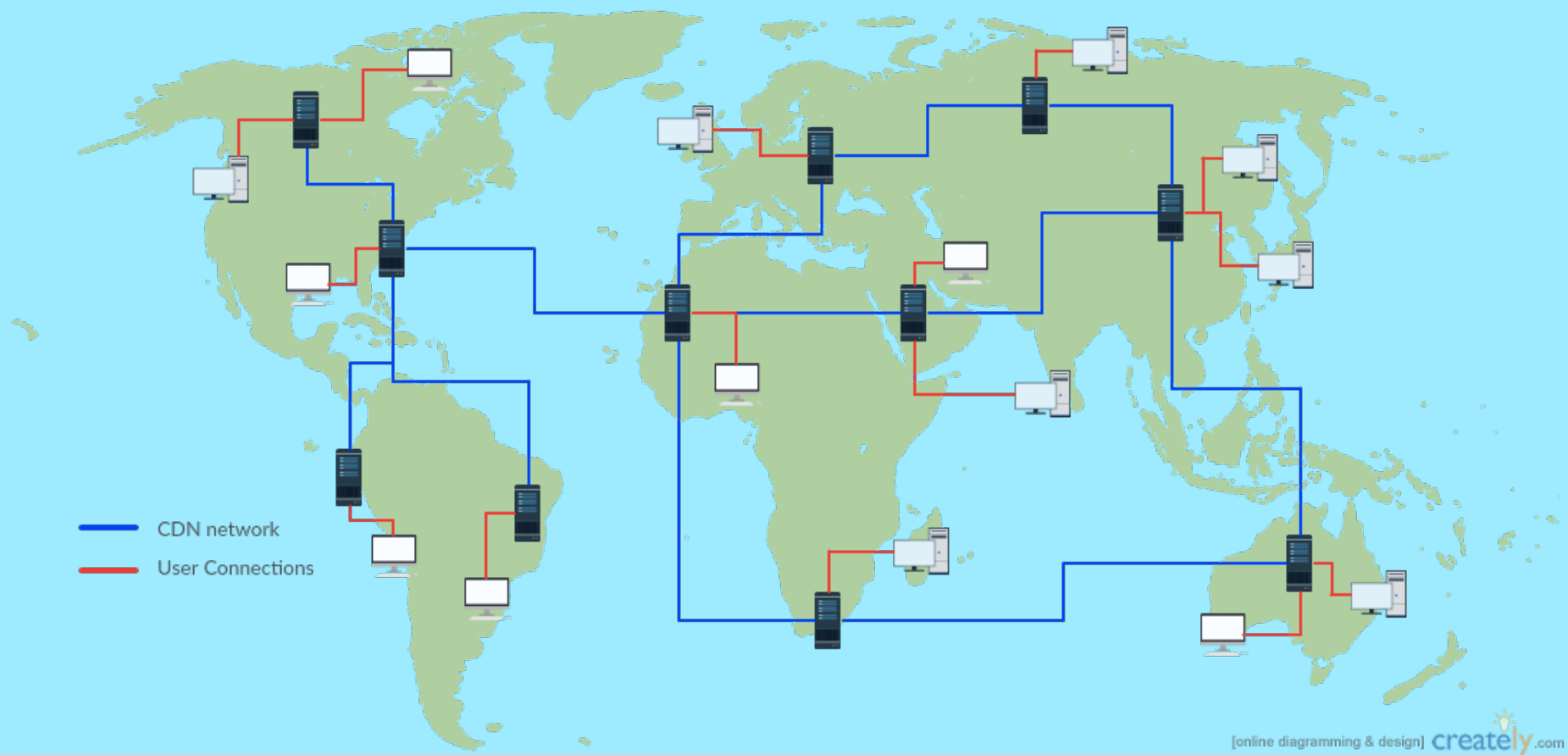


Using Libraries

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.



```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title> Week 11 </title>
6
7   <!-- this script tag links the html script to the jquery api-->
8   <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
9
10  <!-- or you can link to the jquery librari locally. Just be sure your
    • src="FILE PATH" matches your folders/files in your finder -->
11  <!-- <script src="myDirectory/jQuery.js" type="text/javascript"></script>
    • -->
12  <script language="javascript" type="text/javascript"
    • src="theScript.js"></script>
13
14 </head>
15
16 <body>|
17 </body>
18
19 </html>
20
```

[dwnld jquery](#)

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	-	Subtracts 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	&&	exprs1 && 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		exprs1 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"];
```

Keyword	Condition (counter)
<pre>for (var theIndex = 0; theIndex < theYear.length; theIndex ++){ console.log(theYear[theIndex]); }</pre>	<pre>code to execute during loop</pre>

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

```
function buttonClicked() {
```

code statement

```
  console.log("hello");
```

```
}
```

code block inside curly braces

Calling a function

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

Built-in DOM events

Keyboard Events

Event	Description	DOM Level
<u>onkeydown</u>	The event occurs when the user is pressing a key	2
<u>onkeypress</u>	The event occurs when the user presses a key	2
<u>onkeyup</u>	The event occurs when the user releases a key	2

Built-in DOM events

Media Events

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

Built-in DOM properties + methods

Changing HTML Elements

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

Built-in DOM properties + meathods

Adding and Deleting Elements

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

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 and 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 and jQuery can add, remove, and change any of these elements.

JQUERY

JQUERY SLIDES SHARED FROM PROFESSOR LEE TUSMAN

@lee2sman

jQuery is a "Write Less, Do More" JavaScript library. A library is a tool (a collection of simplified commands) used to make it easier to code. It is not a full programming language.

jQuery is a concise JavaScript Library with syntax similar to CSS.

jQuery greatly simplifies JavaScript programming.

jQuery is easy to learn.

HTML is for your content.

CSS is for styling.

jQuery (and Javascript) are used to define behaviors and events.

jQuery is:

Event handling is easier and one method works on all browsers

Once you have made a selection, you can apply multiple methods to it


The basic syntax: `$("selector").action()` - selector can be anything.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
```

You must link with a `<script>` tag to
jQuery on your server or at a CDN's URL **before** you link to
your own script.

typical project layout:


```
project/  
├── css/  
│   └── style.css  
├── js/  
│   └── scripts.js  
└── index.html
```

 projectExample ▶

 index.html

 scripts ▶

 style ▶

 script.js

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 web page "document " is ready (loaded) first.

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.

```
$(document).ready(function() {  
    console.log("jQuery loaded");
```

```
});
```


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.