Javascript Basics

History

1995 — Famously written in one day by Brendan Eich for Netscape in 1995.

2005 — Not taken seriously until 2005 when Google used it to write Google Maps.

2009 — Ryan Dahl writes Node, creating server-side javascript.

2015 — Eric Nylund teaches Yale undergrads & prelims the basics of javascript.

2019 — Taichi Aritomo and Laurel Schwulst teach the basics of JavaScript so that graduate students at Yale can make screensavers

“What language should I learn?”

Why JS

— Built for front-end interaction so perfect for a graphic designer!

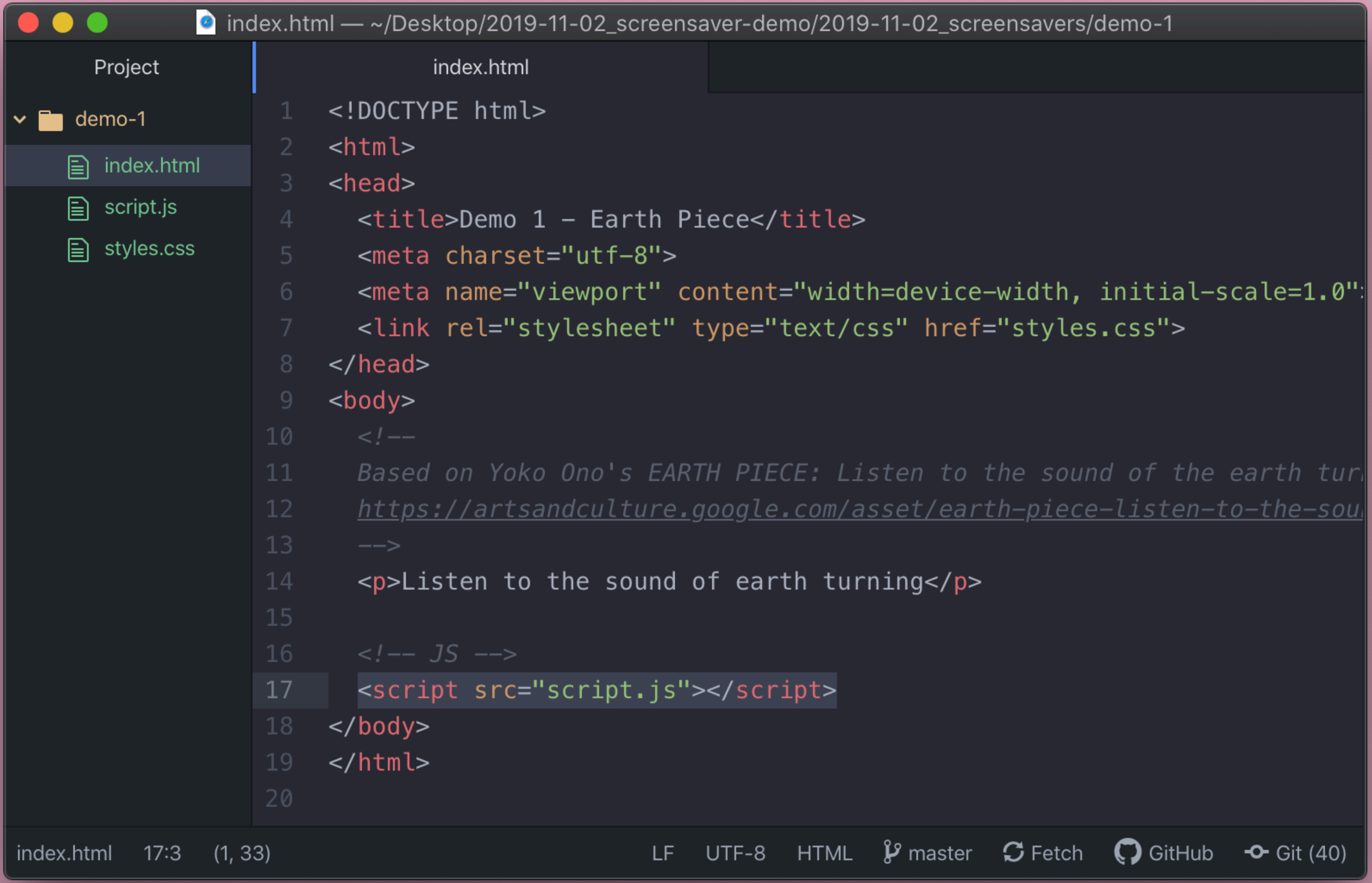
— Also used on the server-side so the syntax will be familiar.

— 100% javascript stack!

— Ever more frameworks use it: jQuery, Node, Mongo Db, Angular.js, etc, etc...

Load a JS file

Best to load it before the closing body tag



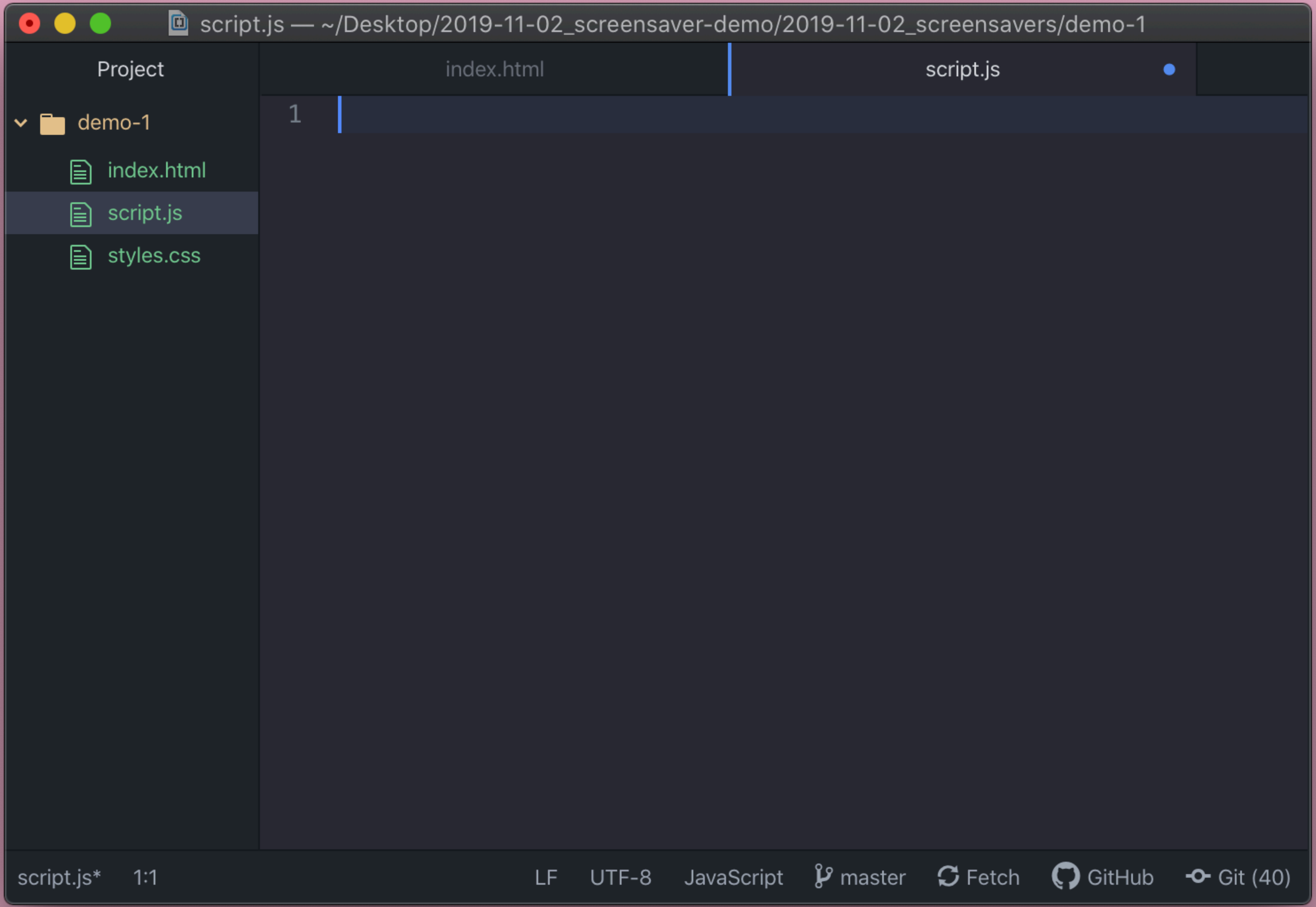
index.html — ~/Desktop/2019-11-02_screensaver-demo/2019-11-02_screensavers/demo-1

Project
demo-1
index.html
script.js
styles.css

```
<!DOCTYPE html>
<html>
<head>
  <title>Demo 1 - Earth Piece</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <!--
  Based on Yoko Ono's EARTH PIECE: Listen to the sound of the earth tur
  https://artsandculture.google.com/asset/earth-piece-listen-to-the-sou
  -->
  <p>Listen to the sound of earth turning</p>

  <!-- JS -->
  <script src="script.js"></script>
</body>
</html>

```

index.html 17:3 (1, 33) LF UTF-8 HTML master Fetch GitHub Git (40)

Writing JS


script.js — ~/Desktop/2019-11-02_screensaver-demo/2019-11-02_screensavers/demo-1

Tip!

Instead of copy-pasting, we encourage you to type out the code you see in these slides, because:

- It’ll help you think more about what you’re writing

- Some of the quote marks in this presentation are “Smart Quotes”, and those will break your code.

Comments

How you and others keep track of what your code does.
The computer ignores it.

You make a single line comment with: //
& a multi-line comment with: /* */

```
// The below function returns all usernames

/*
   The below code is used to get
   the users 10 most recent tweets
*/
```

Debugging

Coding should be done incrementally.

Debugging is a way for you to check
your code as you write it.

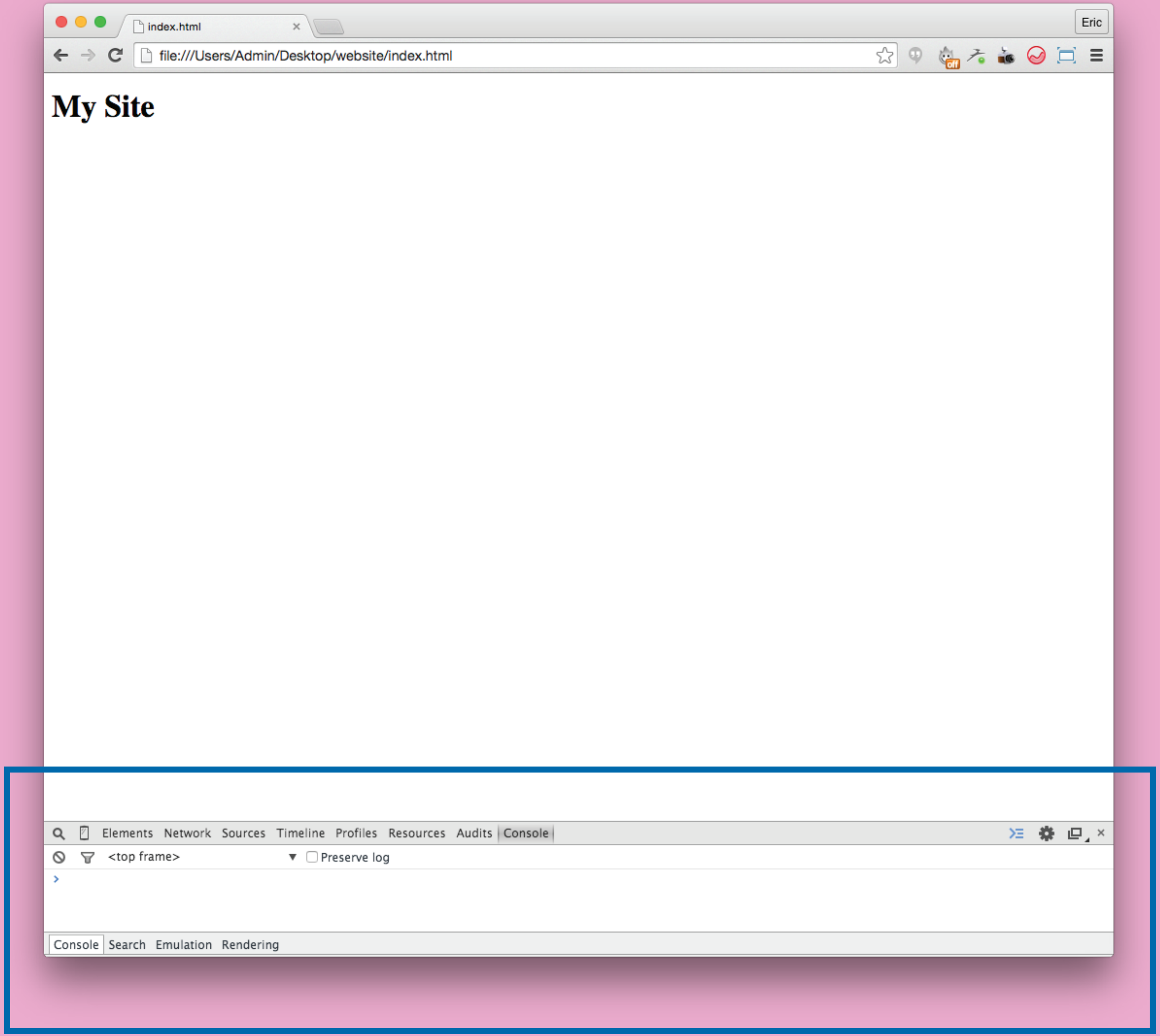
Any time you make a change use `console.log( )`
to be sure you are getting what you expect.

Debugging

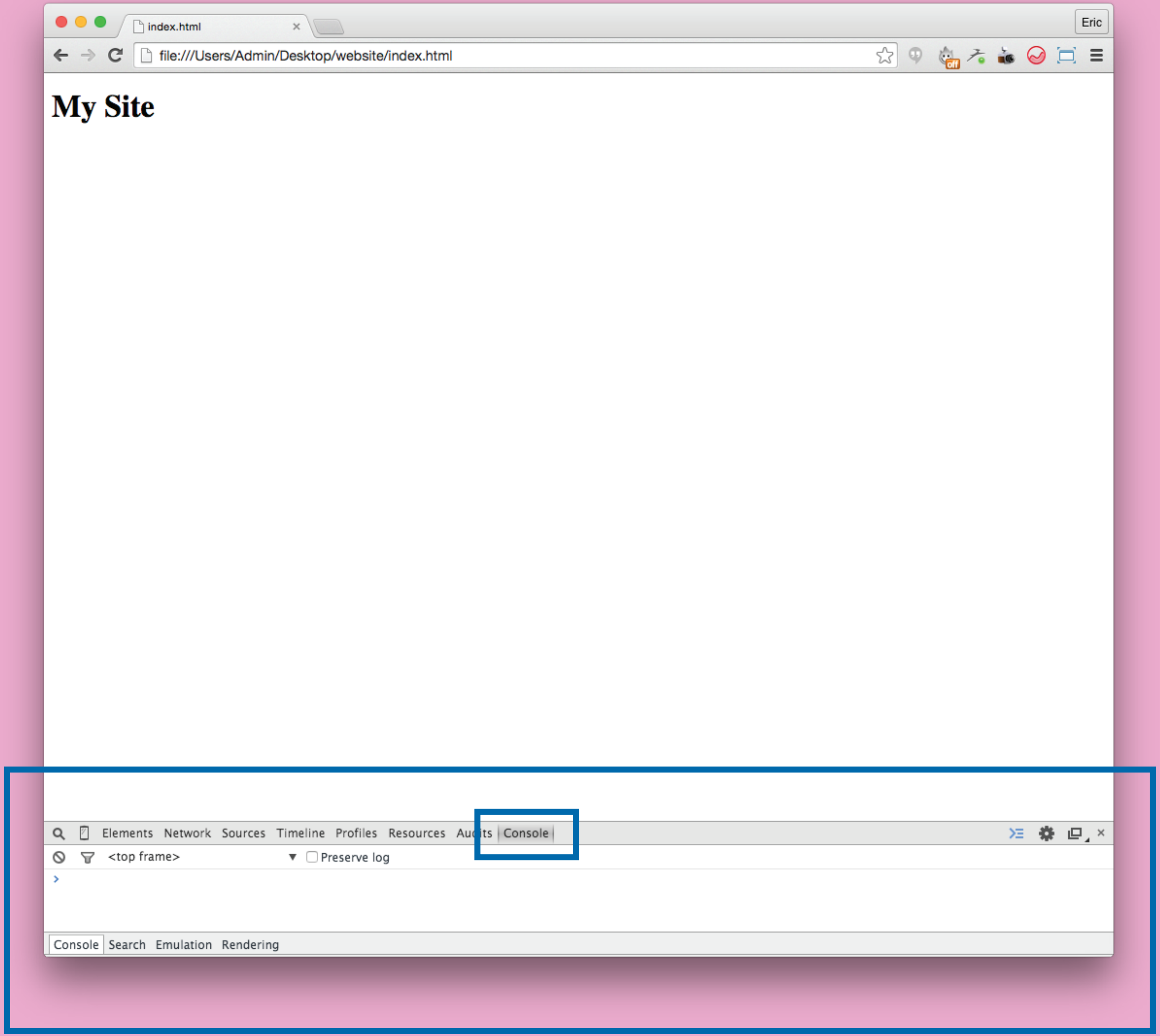
`console.log( )` will take whatever is inside the parentheses and log it to the javascript console in your browser's developer tools.

Most important line of code I'll show you!

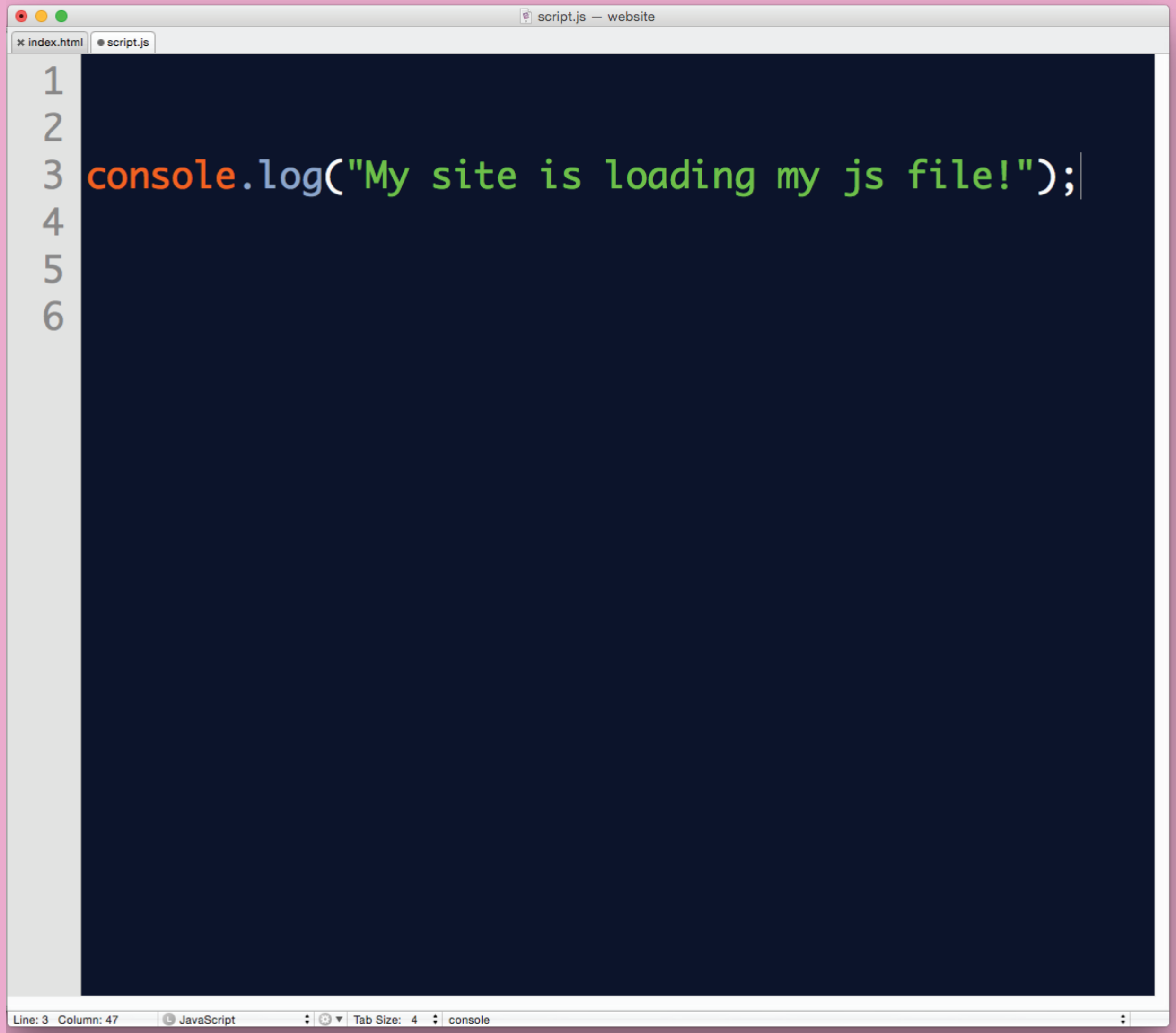
Debugging—Console


index.html
Eric
file:///Users/Admin/Desktop/website/index.html
My Site
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
Preserve log
Console Search Emulation Rendering

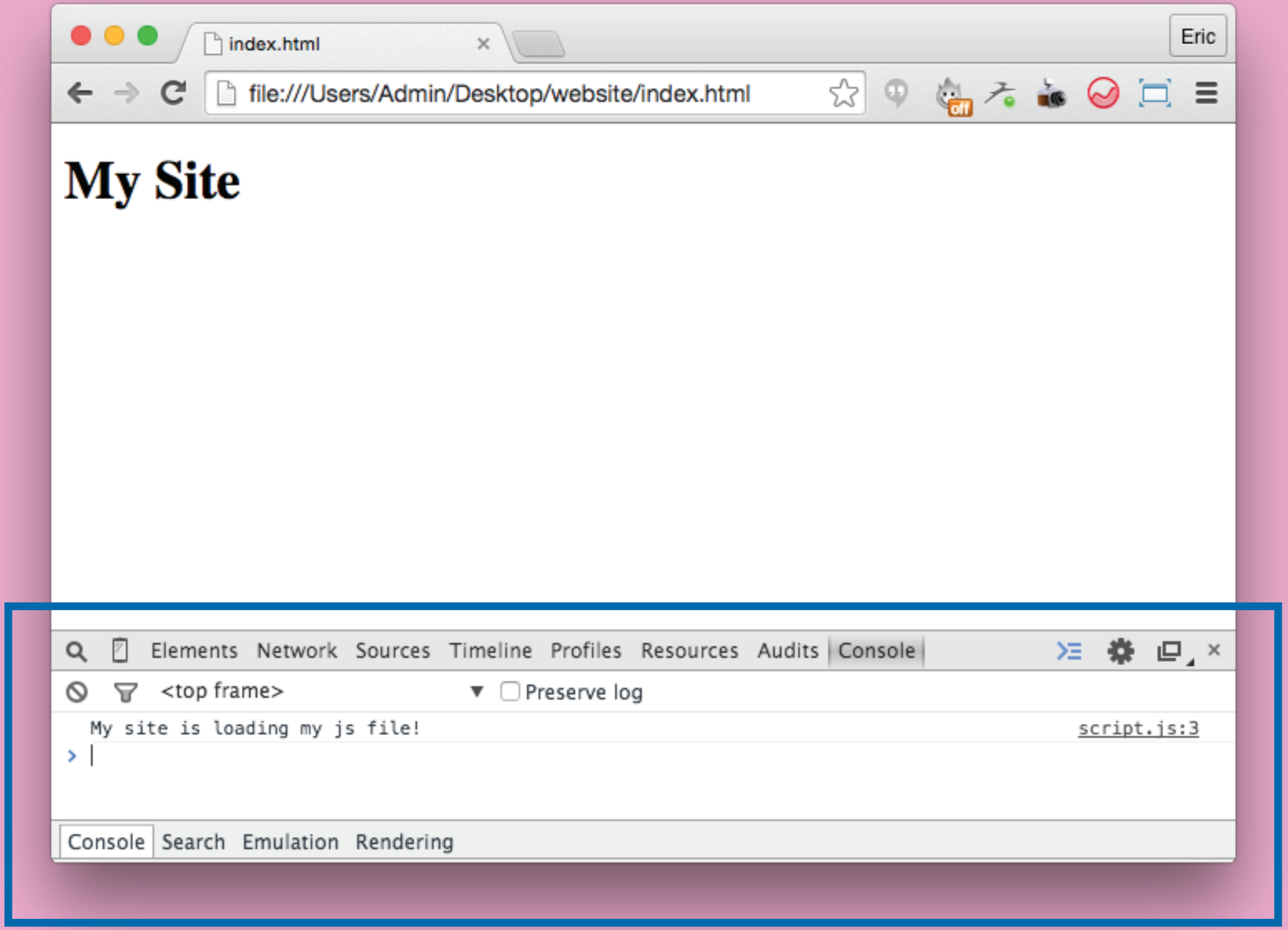
Debugging—Console


index.html
Eric
file:///Users/Admin/Desktop/website/index.html
My Site
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
Preserve log
Console Search Emulation Rendering

Debugging—Example



```
console.log("My site is loading my js file!");
```


index.html
Eric
file:///Users/Admin/Desktop/website/index.html
My Site
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
Preserve log
Console Search Emulation Rendering

```
My site is loading my js file!                                  script.js:3
>
```

Data Types

These are the most basic types of data the language recognizes:

1–Integers
2–Strings
3–Booleans

Data Types—Integers

Used to represent numerical data.

To make a number in your code, just
write a number as numerals without quotes:

```
5 // recent posts to get
190.12334 // div position from top of browser
2*50 // new y-position after each animation
```

Data Types—Strings

Used for storing textual information.

To write a string, surround words with quotes:

```
“eric_nylund” // username
“Interactive Design” // course title
“5” + “7” // makes “57” not 12
```

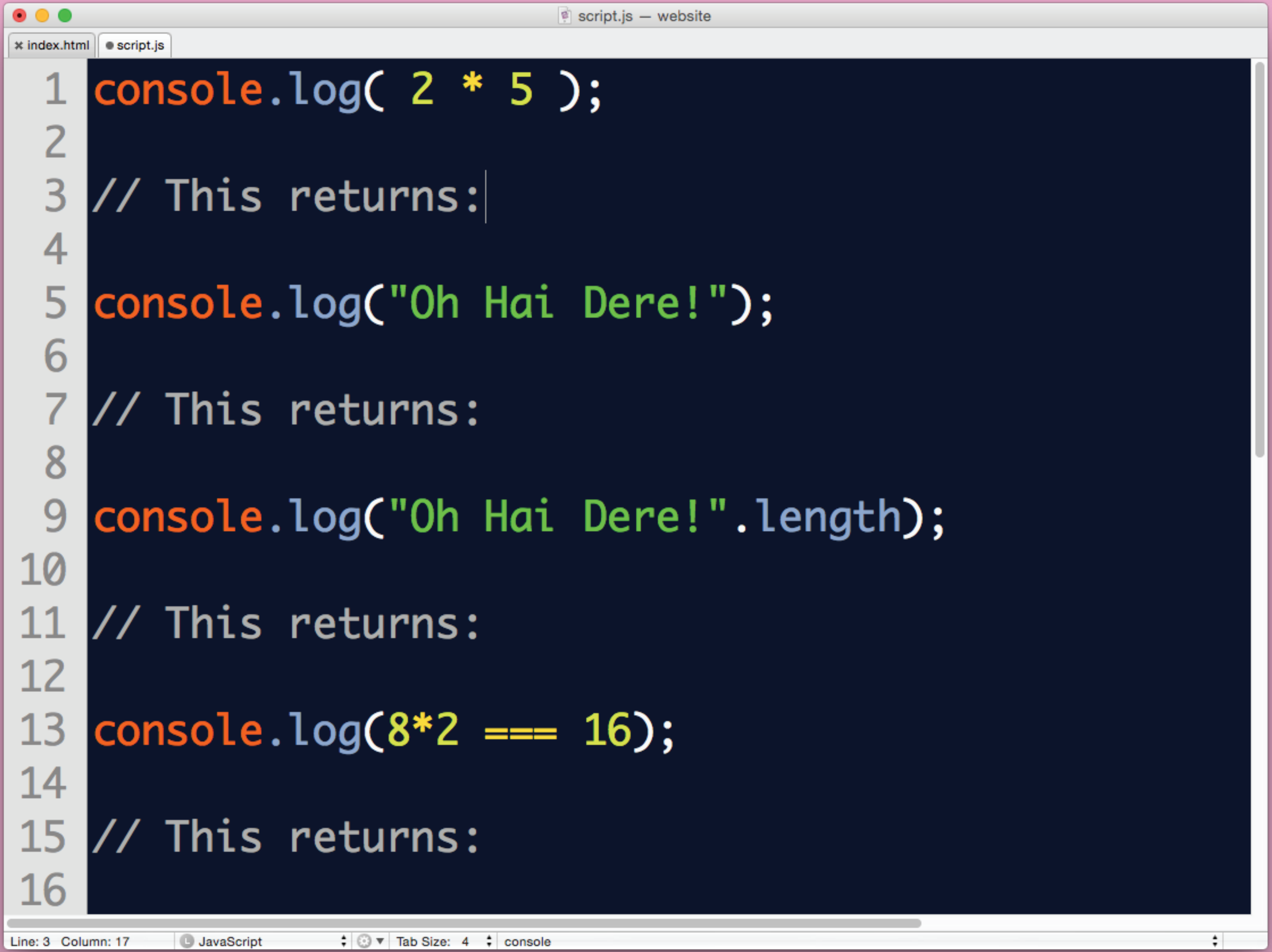
Data Types—Booleans

Used for representing a binary value (true or false):

Often used in comparisons.
Examples:

> 23 > 10 is `true`
> 5 < 4 is `false`
> “abc123” === “abc123” is `true`
> currentStudent is `true`

Data Types—Examples

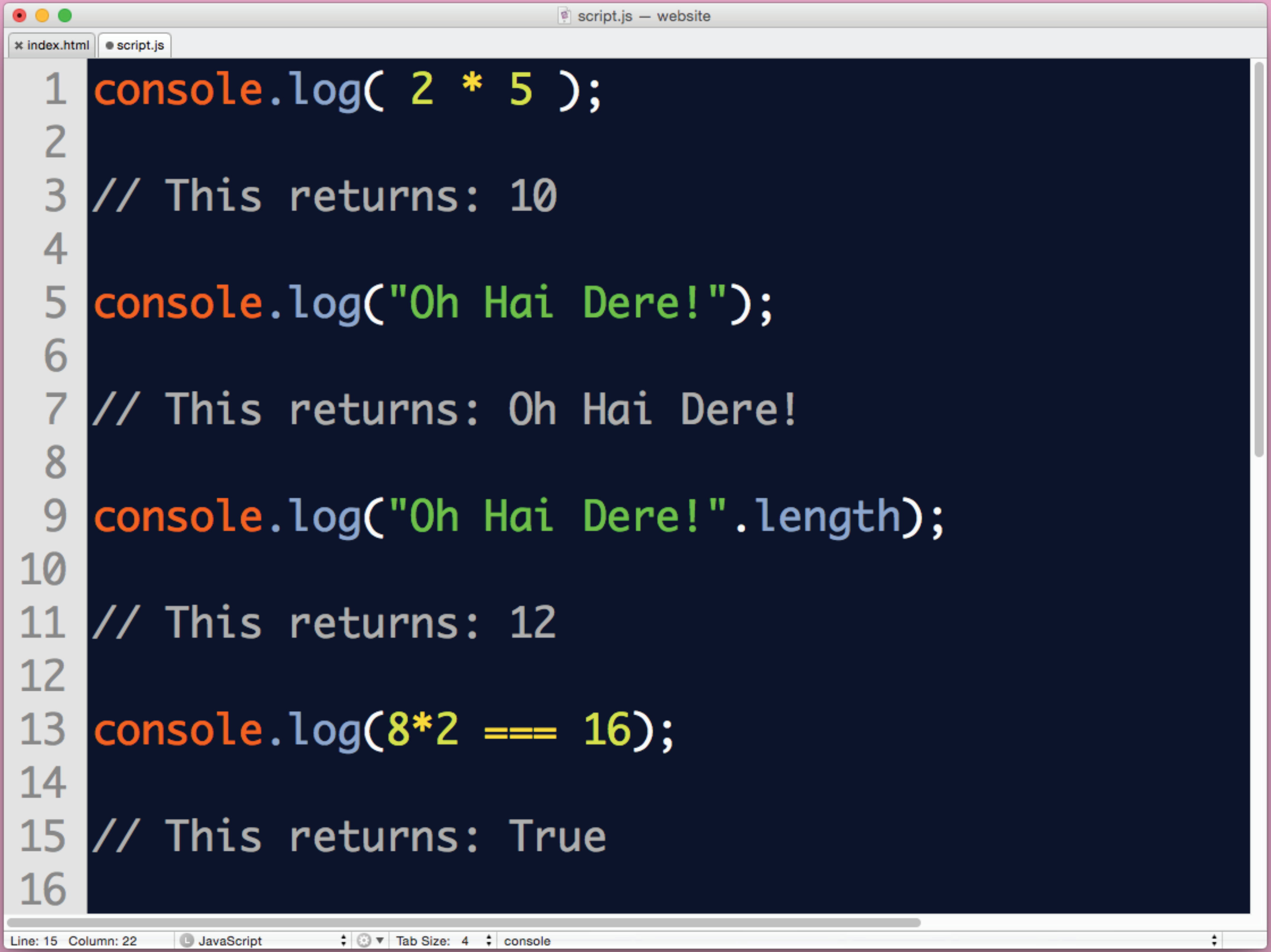

× index.html ● script.js

script.js — website

```
console.log( 2 * 5 );

// This returns:

console.log("Oh Hai Dere!");

// This returns:

console.log("Oh Hai Dere!".length);

// This returns:

console.log(8*2 === 16);

// This returns:

```

Line: 3 Column: 17 JavaScript Tab Size: 4 console

Data Types—Examples


script.js — website

index.html | script.js

```
console.log( 2 * 5 );

// This returns: 10

console.log("Oh Hai Dere!");

// This returns: Oh Hai Dere!

console.log("Oh Hai Dere!".length);

// This returns: 12

console.log(8*2 === 16);

// This returns: True

```

Line: 15 Column: 22 | JavaScript | Tab Size: 4 | console

Variables

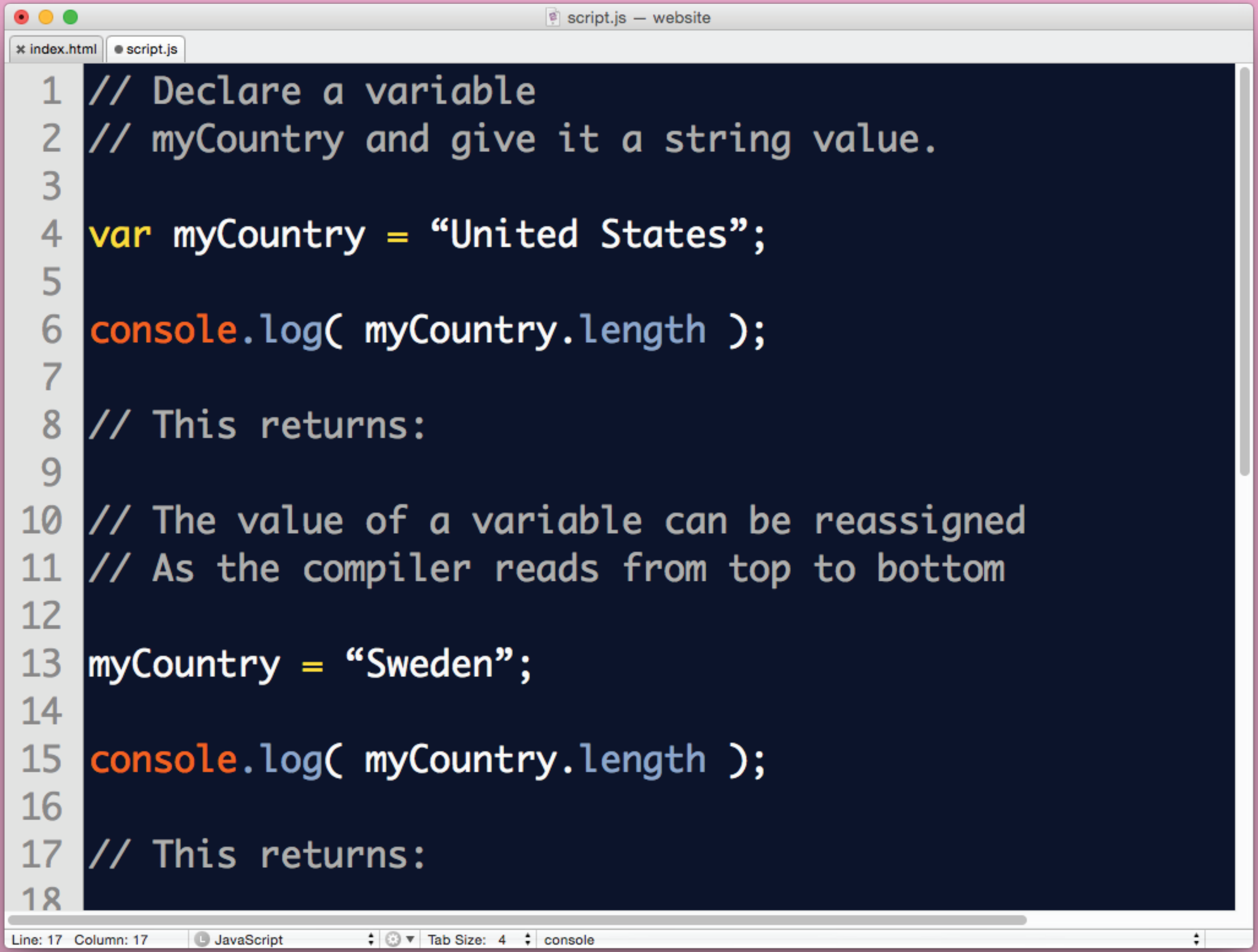
A way to temporarily store values from your coding.

```
    var varName = data;
```

Examples with data types:

```
    var username = “Eric”;
    var classYear = 2015;
    var currentStudent = true;
```

Variables—Examples



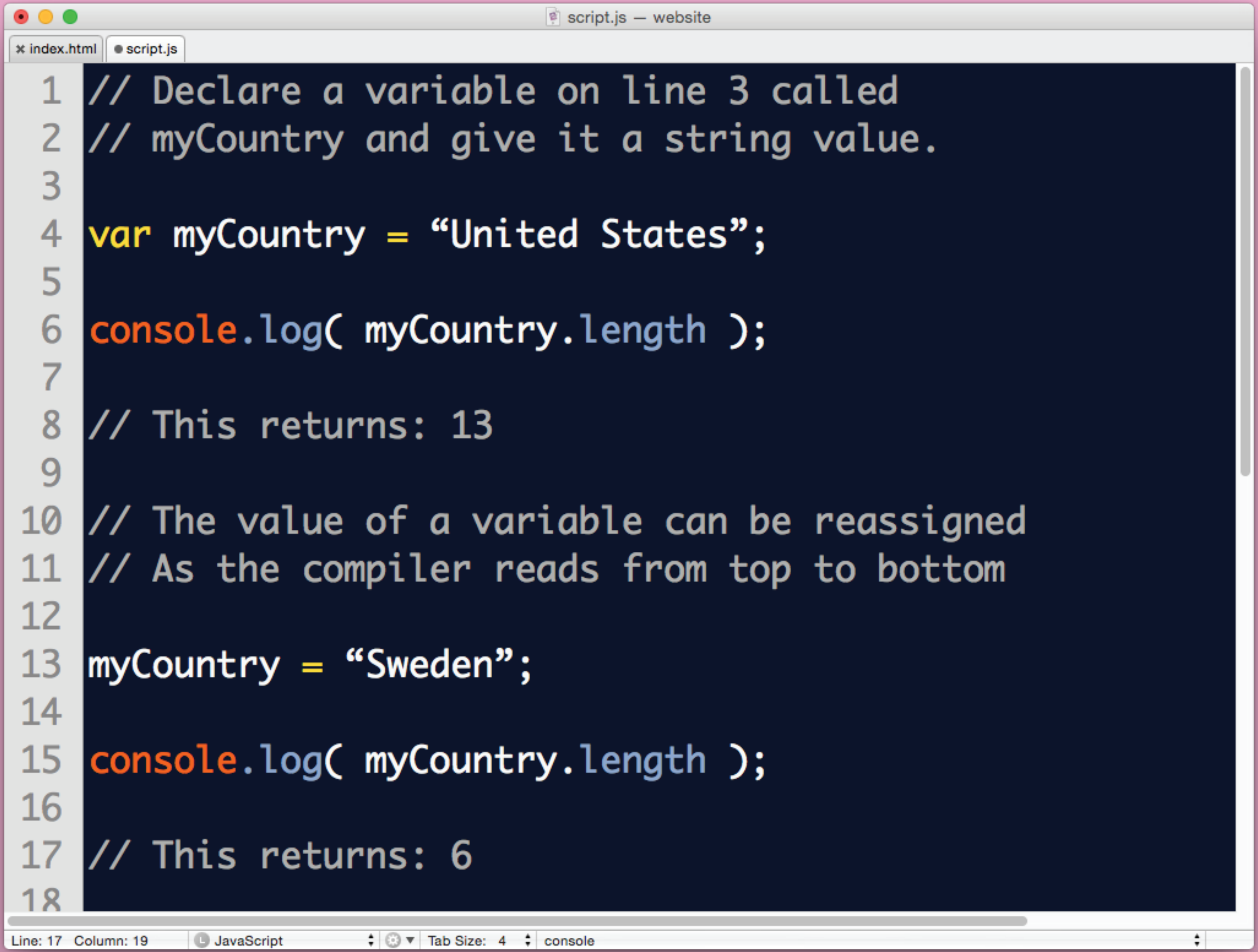
script.js — website

index.html | script.js

```
// Declare a variable
// myCountry and give it a string value.

var myCountry = “United States”;

console.log( myCountry.length );

// This returns:

// The value of a variable can be reassigned
// As the compiler reads from top to bottom

myCountry = “Sweden”;

console.log( myCountry.length );

// This returns:
```

Line: 17 Column: 17 JavaScript Tab Size: 4 console

Variables—Answers



script.js — website

index.html | script.js

```
// Declare a variable on line 3 called
// myCountry and give it a string value.

var myCountry = “United States”;

console.log( myCountry.length );

// This returns: 13

// The value of a variable can be reassigned
// As the compiler reads from top to bottom

myCountry = “Sweden”;

console.log( myCountry.length );

// This returns: 6
```

Line: 17 Column: 19 | JavaScript | Tab Size: 4 | console

Concatenation

A way to add/connect strings using: +

Example:

```
var username = “Eric”;

var greeting = “Hi ” + username + “!”;
```

Questions so far?

Part 2:
Make the computer make decisions! aka Interaction!

Computers think in 1s and 0s.
How can I turn my ideas into
a yes/no logic?

Should a div be visible on load?
Did the user click a button?
Did the user enter the right password?

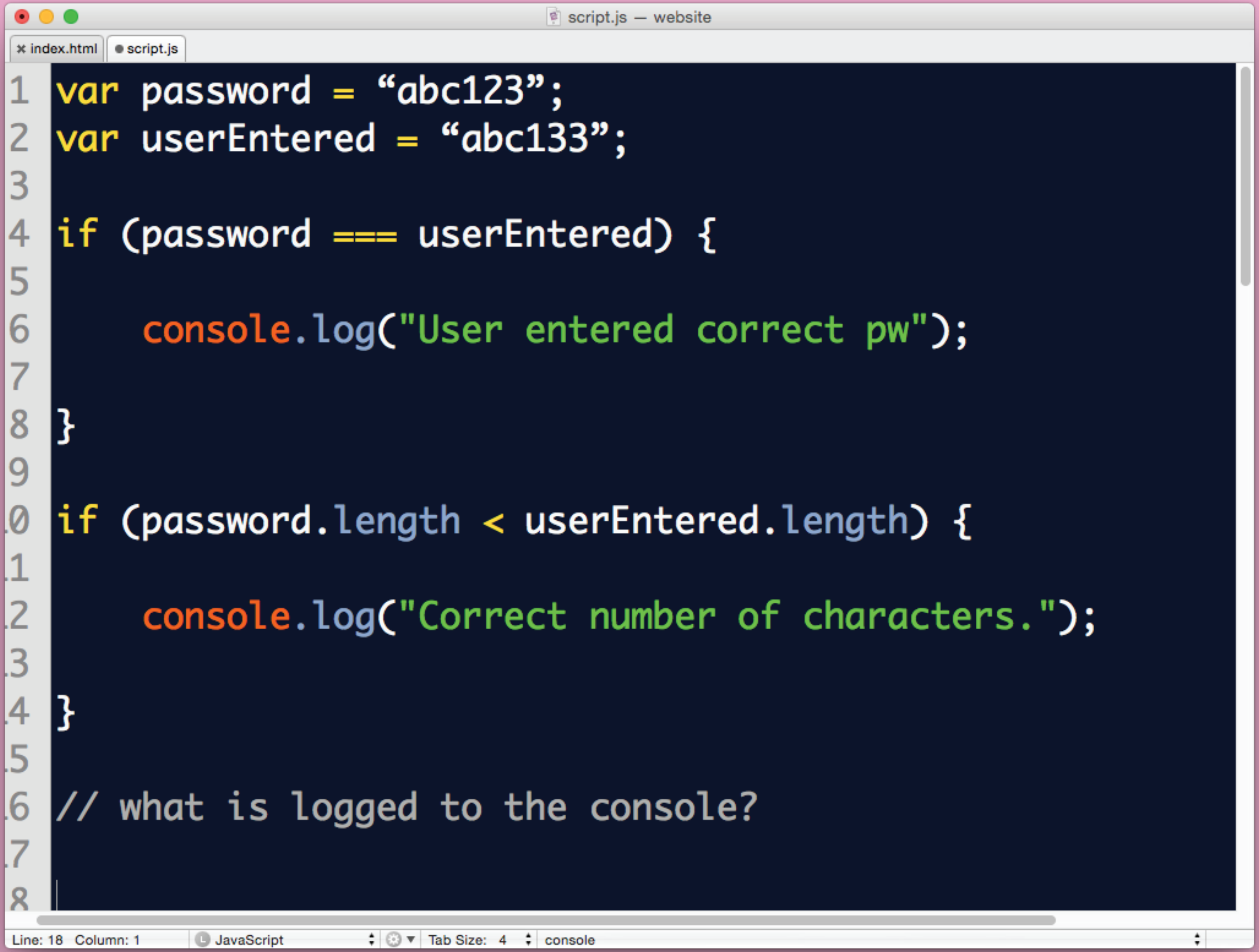
If Statement—Basic yes

Made up of the if keyword, a condition, and a pair of curly braces { }. If the answer to the condition is yes, the code inside the curly braces will run.

Syntax:

```
if ( condition ) {

  // if the condition returns true then
  // execute code inside these brackets.
  // if false, skip this code.

}
```

If Statement—Example



script.js — website

index.html script.js

```
var password = “abc123”;
var userEntered = “abc133”;

if (password === userEntered) {

    console.log("User entered correct pw");

}

if (password.length < userEntered.length) {

    console.log("Correct number of characters.");

}

// what is logged to the console?

```

Line: 18 Column: 1 JavaScript Tab Size: 4 console

If/Else Statement—Basic yes/no

In addition to doing something when the condition is true, we can do something else if the condition is false.

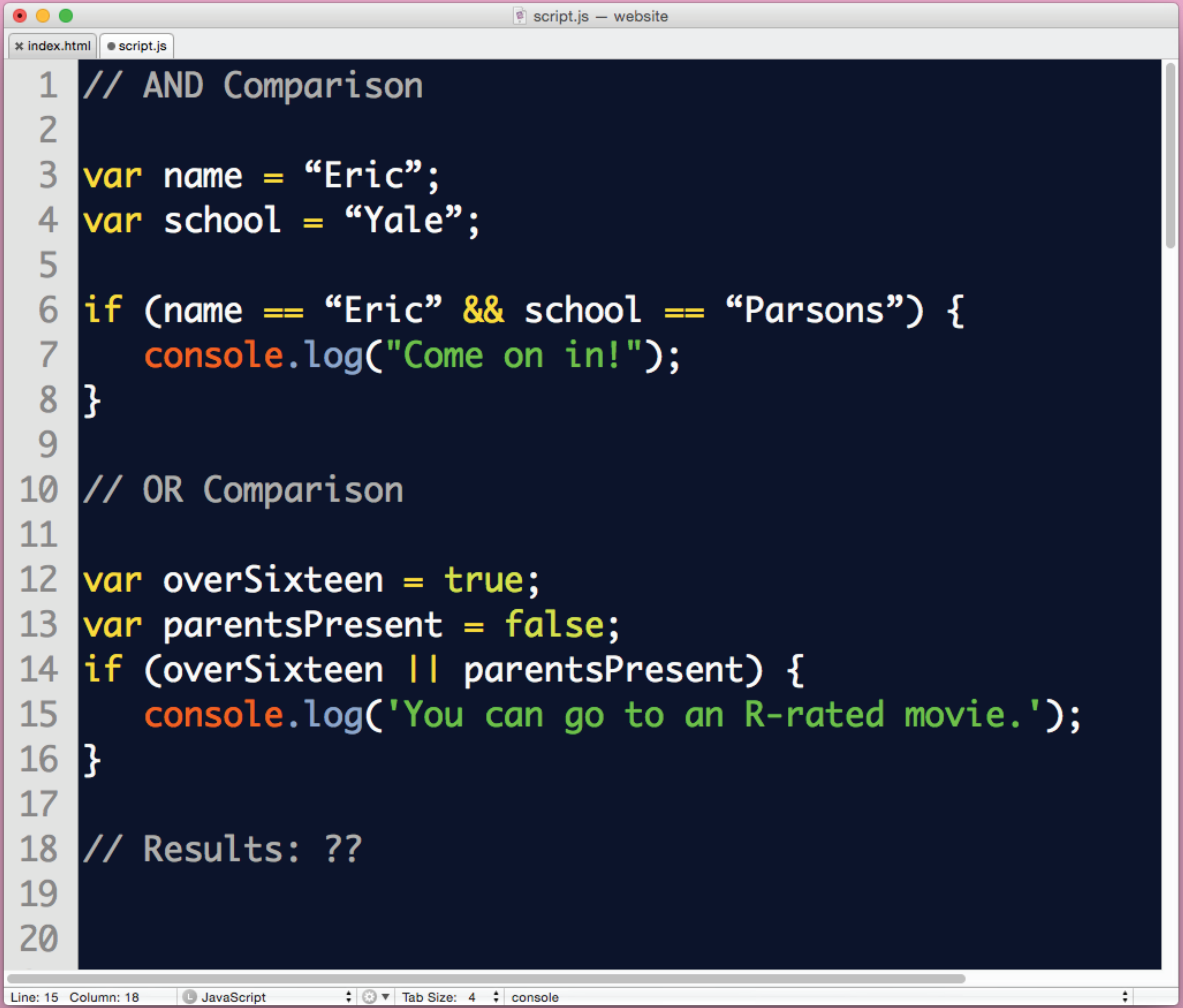
```
if ( currentStudent === true ) {
  console.log(“You are currently enrolled”);
}
else {
  console.log(“You are not enrolled”);
}
```

Logical Operators

Used when several conditions need to be evaluated at once.

&& (and) || (or)   !== (not equal)

Logical Operators – Examples



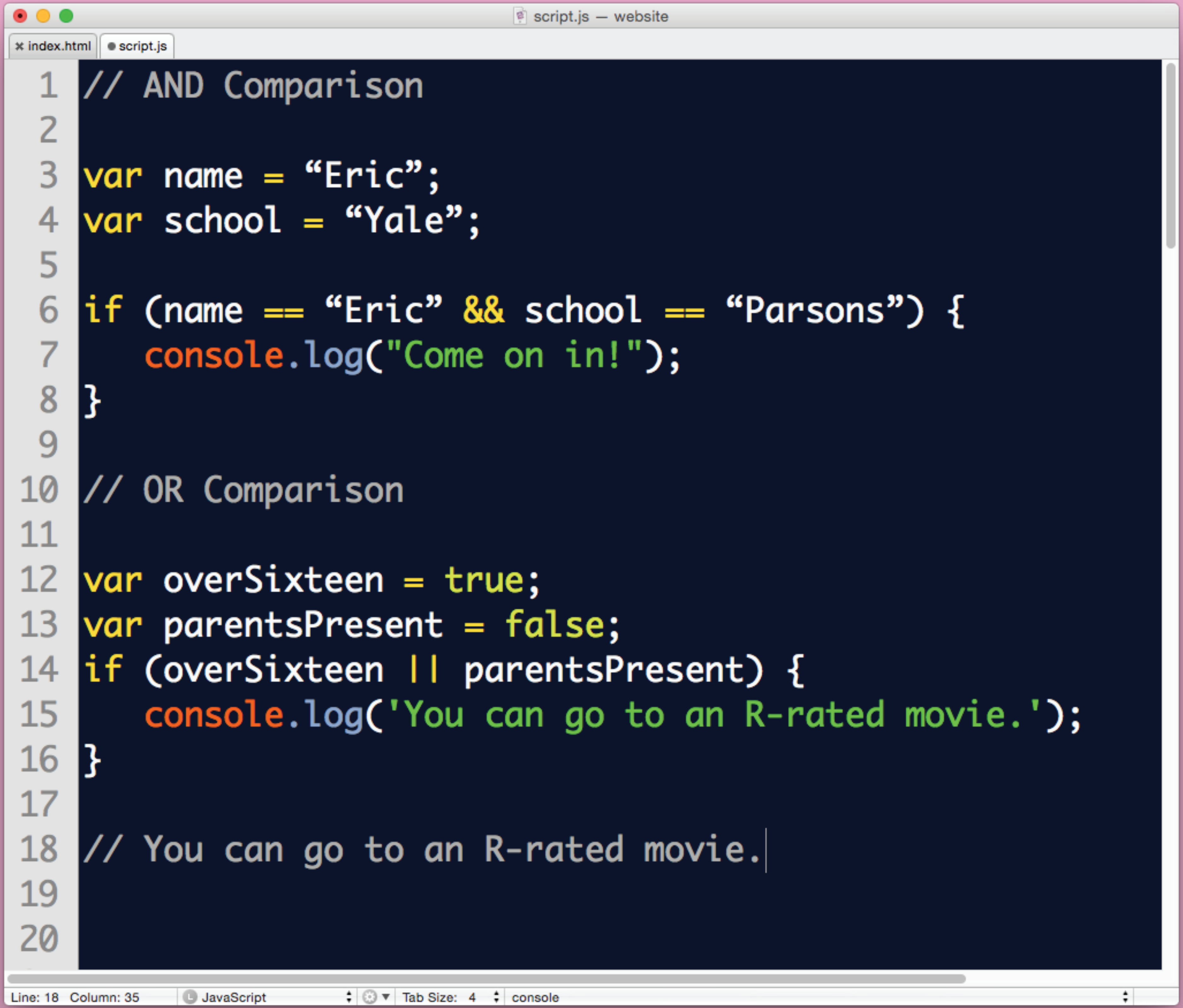
script.js — website

index.html | script.js

```
// AND Comparison

var name = “Eric”;
var school = “Yale”;

if (name == “Eric” && school == “Parsons”) {
    console.log("Come on in!");
}

// OR Comparison

var overSixteen = true;
var parentsPresent = false;
if (overSixteen || parentsPresent) {
    console.log('You can go to an R-rated movie.');
}

// Results: ??


```

Line: 15 Column: 18 | JavaScript | Tab Size: 4 | console

Logical Operators – Answers



script.js — website

index.html | script.js

```
// AND Comparison

var name = “Eric”;
var school = “Yale”;

if (name == “Eric” && school == “Parsons”) {
    console.log("Come on in!");
}

// OR Comparison

var overSixteen = true;
var parentsPresent = false;
if (overSixteen || parentsPresent) {
    console.log('You can go to an R-rated movie.');
}

// You can go to an R-rated movie.
```

Line: 18 Column: 35 | JavaScript | Tab Size: 4 | console

Arrays

So far, we've only been able to store ONE number or ONE string.

Arrays are used to store a set of related things.

```
var arrayName = [data, data, data];

var studentInfo = [“Eric”, “Yale”, 2015];
```

Arrays—Access Elements

```
var studentInfo = [ “Eric”, “Yale”, 2015 ];
                      0       1      2
```

Arrays—Example

```
var studentInfo = [ “Eric”,  “Yale”,  2015  ];

console.log( studentInfo[0] );

//Result: ??
```

Arrays—Example

```
var studentInfo = [ “Eric”,  “Yale”,  2015  ];

console.log( studentInfo[0] );

//Result: Eric
```

While Loop

Repeat a block of code as long as a condition is still true

```
while ( condition ) {
  // code to execute
}
```

While Loop — Example

```
var counter = 1;

while ( counter < 7 ) {

  console.log(counter);
  counter++;

}

// Result: ?
```

While Loop — Example

```
var counter = 1;

while ( counter < 7 ) {

  console.log(counter);
  counter++;

}

// Result: 123456
```

While Loop — Example

```
var counter = 1;

while ( counter < 7 ) {

    counter++;
    console.log(counter);

}

// Result: ?
```

While Loop — Example

```
var counter = 1;

while ( counter < 7 ) {

  counter++;
  console.log(counter);

}

// Result: 234567
```

For Loop

Repeat a block of code a set number of times. Most often used to access elements of an array one by one.

```
for (counter; condition; increment) {

  //Code to execute

}
```

For Loop

Repeat a block of code a set number of times. Most often used to access elements of an array one by one.

```
for (counter; condition; increment) {

  //Code to execute

}
```

Decides how many times the loop will run

For Loop—Example

```
for (var counter = 1; counter < 7; counter++) {

  console.log(counter);

}


//Result:?
```

For Loop—Example

```
for (var counter = 1; counter < 7; counter++) {
  console.log(counter);
}

//Result:?
```

For Loop—Example

```
for (var counter = 1; counter < 7; counter++) {

}

//Result:?
```

Set before first loop

Checked before every loop

Done at the end of each loop

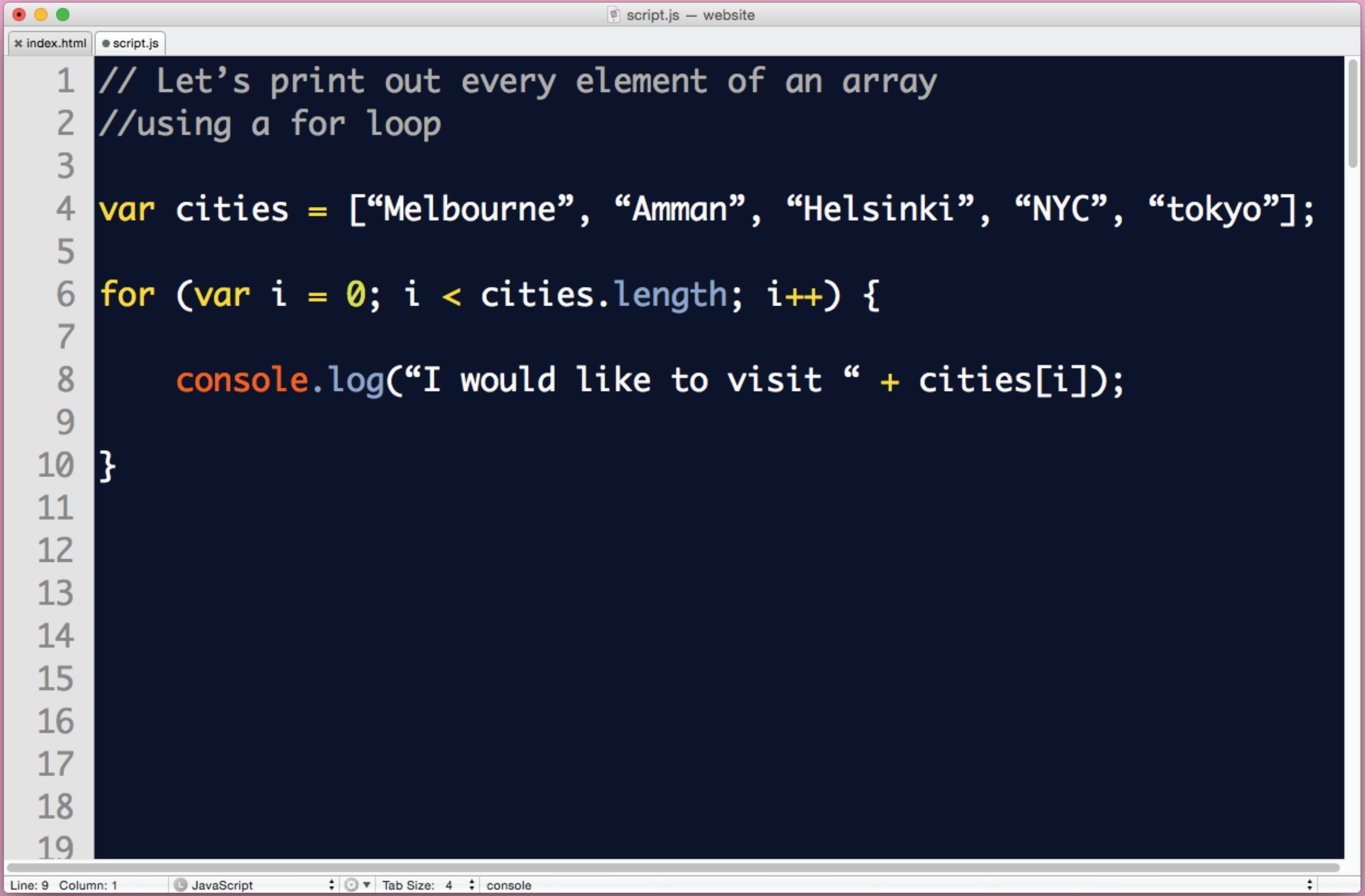
For Loop—Example

```
for (var counter = 1; counter < 7; counter++) {
  console.log(counter);
}

//Result:?
```

For Loop—Answer

```
for (var counter = 1; counter < 7; counter++) {
  console.log(counter);
}

//Result:123456
```

For Loop w/ Array



```
// Let’s print out every element of an array
//using a for loop

var cities = [“Melbourne”, “Amman”, “Helsinki”, “NYC”, “tokyo”];

for (var i = 0; i < cities.length; i++) {

    console.log(“I would like to visit “ + cities[i]);

}
```

For Loop and While Loop — Comparison

Two ways to do the same thing

```
for (var counter = 1; counter < 7; counter++) {
  console.log(counter);
}


var counter = 1;
while (counter < 7) {
  console.log(counter);
  counter++;
}
```

For Loop and While Loop — Comparison

```
for (var counter = 1; counter < 7; counter++) {
  console.log(counter);
}
```

For — Good when you know how many iterations you're going to have

```
var counter = 1;
while (counter < 7) {
  console.log(counter);
  counter++;
}
```

While — Good when you don't know how many iterations you're going to have

When is While Loop useful? — Example

While loops are best when you don’t know how many iterations you’re going to have. e.g. you’re rolling a die but you really just want a 6.

```
// get a random number from 1 to 6
var number = Math.ceil( Math.random()*6 );

// keep trying as long as the number isn’t 6
while (number != 6) {
  number = Math.ceil( Math.random()*6 );
  console.log(number);
}
```

When is While Loop useful? — Example

While loops are best when you don’t know how many iterations you’re going to have. e.g. you want to roll a die but you don’t want to roll a 3.

```
var myRoll = 3;
var numberIDontWant = 3;

while (myRoll == numberIDontWant) {
  myRoll = Math.random() * 6; // random num from 0 to 5.9999…
  myRoll = Math.ceil( myRoll ); // round the number up
  console.log(myRoll);
}
```

Functions

Don’t Repeat Yourself (D.R.Y)

The D.R.Y. principle is really important in programming. No repeating!

Any time you find yourself typing the same thing, but modifying only one small part, you can probably use a function.

Functions—Syntax

First define the function:

```
var functionName = function( variable ) {
    // code code code
    // code code code
    // (more lines of code)
};
```

Functions—Syntax

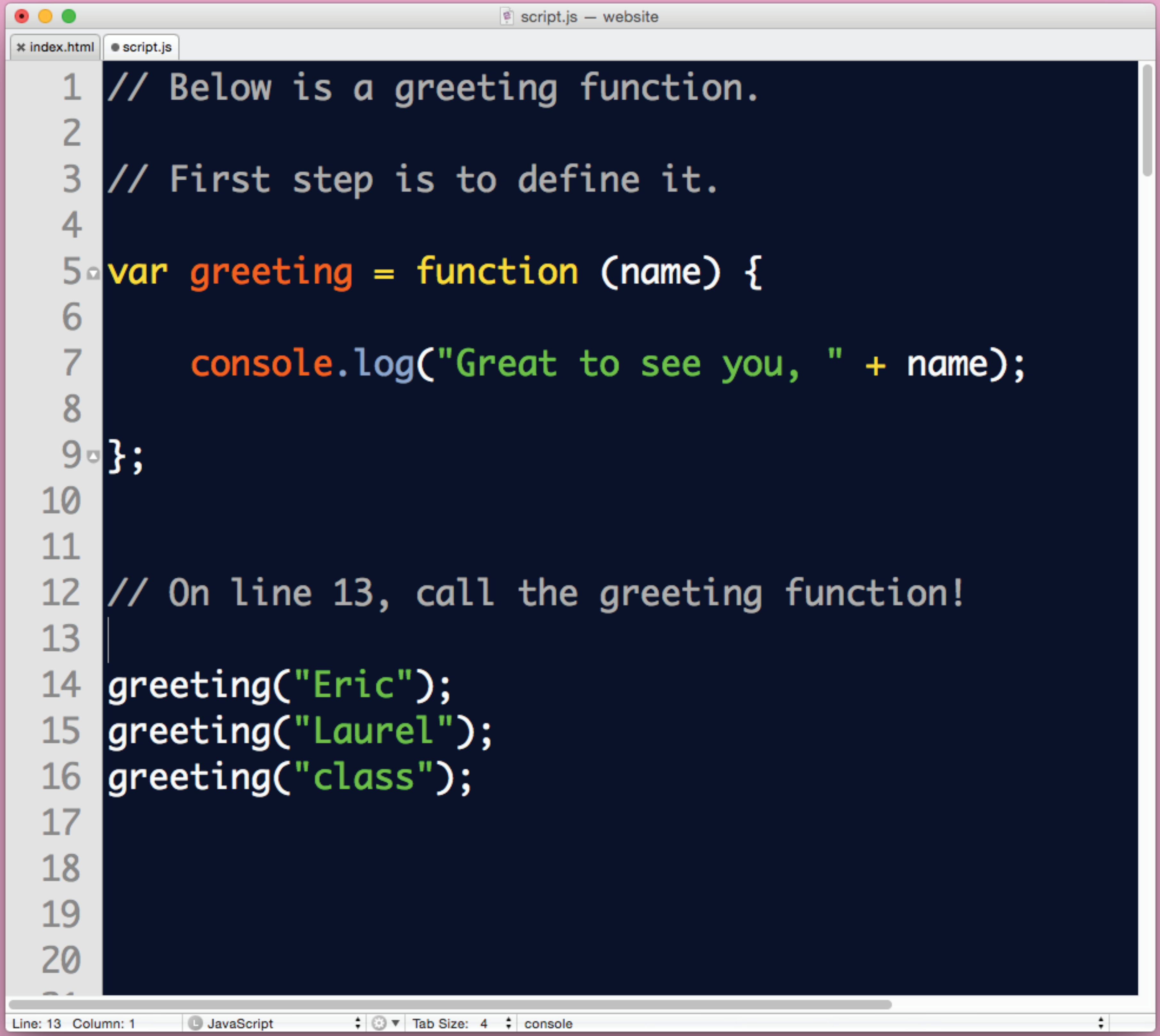
First define the function:

```
var functionName = function( variable ) {
    // code code code
    // code code code
    // (more lines of code)
};
```

Then you can call it at any time:

```
functionName(value1);
functionName(value2);
```

Functions—Example



```
 1  // Below is a greeting function.
 2
 3  // First step is to define it.
 4
 5  var greeting = function (name) {
 6
 7      console.log("Great to see you, " + name);
 8
 9  };
10
11
12  // On line 13, call the greeting function!
13
14  greeting("Eric");
15  greeting("Laurel");
16  greeting("class");
17
18
19
20
```

Line: 13 Column: 1 JavaScript Tab Size: 4 console

Function Syntax (Alternative)

You can also define and call a function without storing it in a variable…

```
function functionName( input ) {
  // code code code
  // code code code
}

functionName(value1);
functionName(value2);
```

Timing

Run a function after 1000 milliseconds (1 second)

```
function functionName() {
  // code code code
  // code code code
}

// wait 1000 milliseconds, then call functionName
setTimeout( functionName, 1000 );
```

Timing

Run a function every second, forever…

```
function tick() {
  console.log(“tick”);
  setTimeout(tick, 1000); // schedule next tick
}

tick(); // call tick for the first time
```

Variable Scope—Local vs Global

When you define a variable inside brackets,
it only exists inside the brackets!

Local Variable:

```
var bar = function() {
  var localVar = “howdy”;
}
console.log(localVar);  // error
```

DOM Element

DOM Element is a special data-type that lets you interact with HTML elements.

In your HTML:

```
<div id="special">my special element</div>
<div id="regular">my regular element</div>
```

In your Javascript:

```
var myElement = document.querySelector("#special");
console.log(myElement); // prints the DOM Element
```

DOM Element

document.querySelector() lets you grab the first DOM Element in your HTML that matches the given CSS selector.

In your HTML:

```
<div class="boxes">
  <div class="box">first box</div>
  <div class="box">second box</div>
</div>
```

In your Javascript:

```
var myElement = document.querySelector(".box");
console.log(myElement); // prints the first box
```

DOM Element

One way to be very specific is to get fancy with CSS selectors

In your HTML:

```
<div class="boxes">
  <div class="box">first box</div>
  <div class="box">second box</div>
</div>
```

In your Javascript:

```
var myElement = document.querySelector(".box:nth-child(2)");
console.log(myElement); // prints the second box
```

DOM Element(s)

document.querySelectorAll() gives you a collection of all the elements that match the given CSS Selector.

In your HTML:

```
<div class="box">first box</div>
<div class="box">second box</div>
```

In your Javascript:

```
var myElements = document.querySelectorAll(".box");
console.log(myElements);     // prints all the boxes
console.log(myElements[0]); // prints the first box
```

DOM Element — Properties and Methods

DOM Elements come with built-in properties and methods to interact with the element on the page. Here are some useful ones!

Javascript:

```
    var div = document.querySelector(“div”);

    div.innerHTML = “<p>Hey!</p>”; // change the HTML
    div.setAttribute(“id”, “greeting”); // set an attribute
    div.className = “big”; // set a CSS class
    div.classList.add(“visible”); // add a CSS class
    div.classList.remove(“visible”); // remove a CSS class
    div.style[“width”] = “100px”; // set a CSS property
```

Variable Scope—Local vs Global

When you define a variable inside brackets,
it only exists inside the brackets!

Local Variable:

```
    var bar = function() {
      var localVar = “howdy”;
    }
    console.log(localVar);  // error
```

Going Further: Frameworks

A lot of frameworks have been built on top of Javascript. They can save you the work of writing and testing complicated systems, but can make it harder to break their rules or assumptions.

React

- For web apps that listen and respond to real-time changes in data

Gatsby

- For making static websites using reusable React-based templates and components

Javascript

The programming language of HTML and the Web. Interaction with the user, animation, etc, all done with javascript.

jQuery

Javascript library designed to simplify the client-side scripting of HTML.

Load a JS file

Order matters!!

```
<!DOCTYPE html>
<html>
  <head>
      <title></title>

  </head>
  <body>

      <div></div>
  <script type='text/javascript' src='js/jquery.min.js'></script>
  <script type='text/javascript' src='js/script.js'></script>
  </body>
</html>
```

Resources

w3schools
Good quick reference with simple examples
- Javascript portal

Mozilla Developer Network (MDN)
Most up-to-date/legit documentation for web-standard HTML/CSS/JS
- Javascript portal
- In-depth beginner's tutorial

Final Notes

Google is your best friend!
stackoverflow.com

All this is gone over on:
codeacdemy.com

eloquentjavascript.net/