Data types, variables, loops, functions and objects

Lesson 2

Plan for this lesson

Data types

- Code along

Variables

- Code along

Loops

- Code along

Functions

- Code along

Objects

- Code along

Data types

JavaScript Data Types Repetition and detail

We already looked at data types in JavaScript last lesson, but we’ll look a bit more at the concept in detail here.

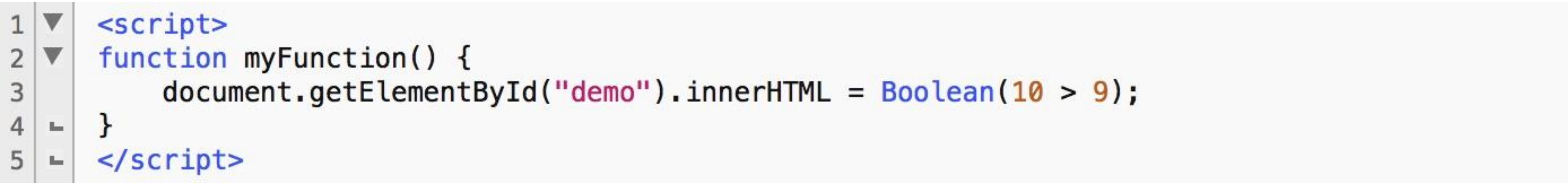
There are six simple data types:

- Boolean
- Null
- Undefined
- Number
- String
- Symbol

And of course objects, which are far from simple and which we’ll get back to.

JavaScript Data Types Boolean

In computer science, a *boolean* is a logical data type that can have only the values true or false.



```
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = Boolean(10 > 9);
}
</script>
```

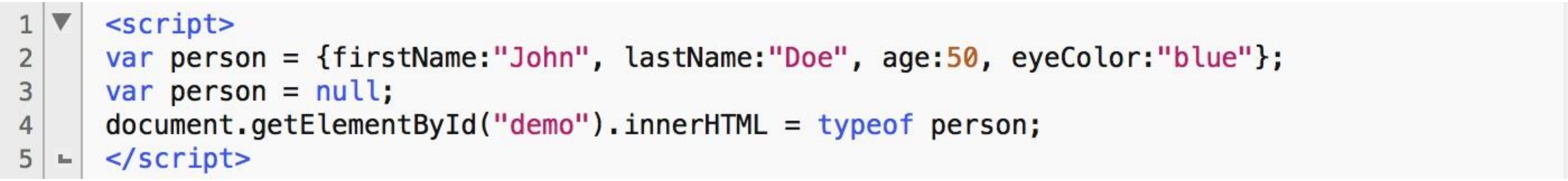
JavaScript Data Types Null

In computer science, a *null* value represents a reference that points, generally intentionally, to a nonexistent or invalid object or address. The meaning of a null reference varies among language implementations.

Unfortunately, in JavaScript, the data type of null is an object.

> *Some consider it a bug in JavaScript that the type of null is an object.*
> *It should be null.*

You can empty an object by setting it to null:

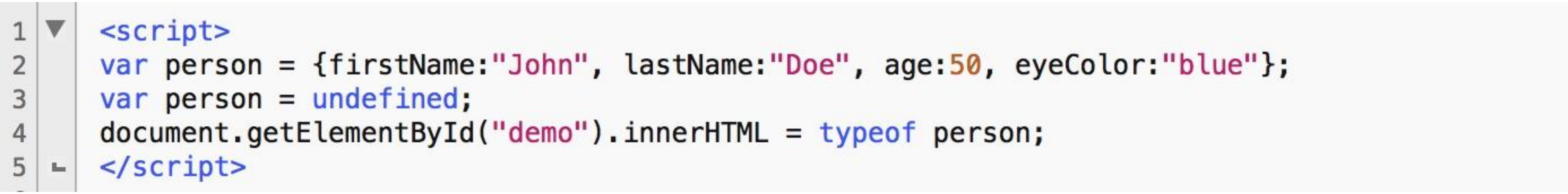


```
<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
var person = null;
document.getElementById("demo").innerHTML = typeof person;
</script>
```

JavaScript Data Types Undefined

A primitive value automatically assigned to variables that have just been declared or to formal arguments for which there are no actual arguments.

You can also empty an object by setting it to undefined:



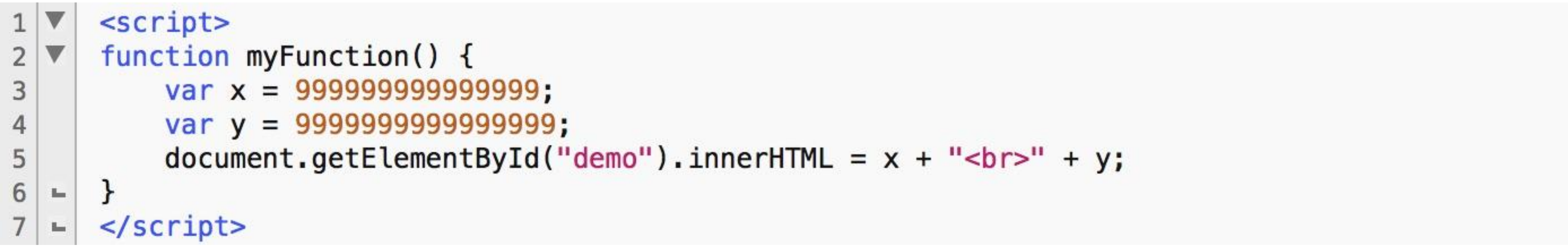
```
<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
var person = undefined;
document.getElementById("demo").innerHTML = typeof person;
</script>
```

There are some differences between *undefined* and *null* though:

- *undefined* is undefined, which is literally nothing
- *null* is an object containing nothing

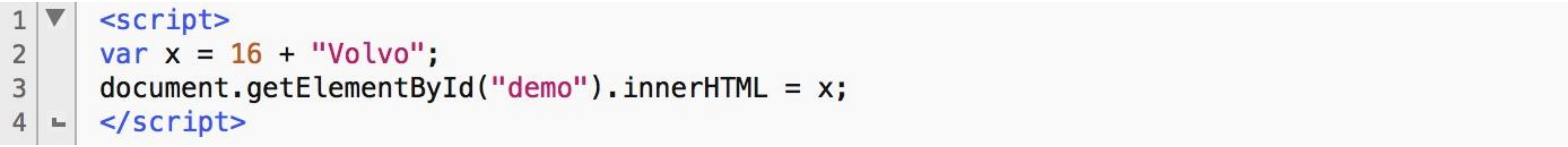
JavaScript Data Types Number

In JavaScript, *Number* is a numeric data type in the double-precision floating point format. In other programming languages different numeric types can exist, for examples: Integers, Floats, Doubles, or Bignums.



```
<script>
function myFunction() {
    var x = 999999999999999;
    var y = 9999999999999999;
    document.getElementById("demo").innerHTML = x + "<br>" + y;
}
</script>
```

When adding a number and a string, JavaScript will treat the number as a string.



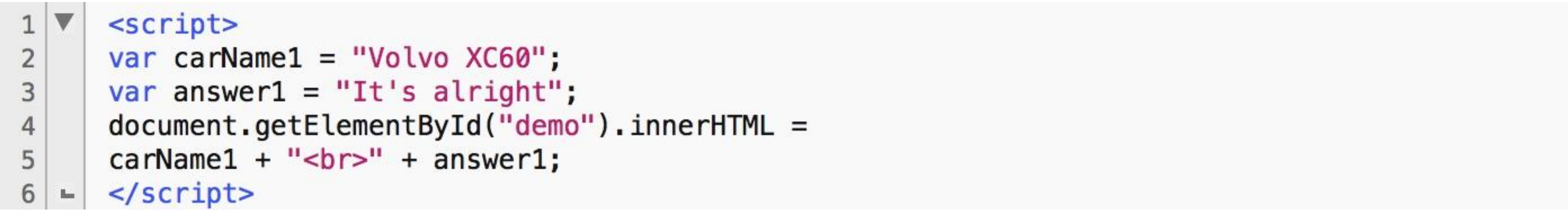
```
<script>
var x = 16 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>
```

JavaScript Data Types String

In any computer programming language, a string is a sequence of characters used to represent text.

In JavaScript, a *String* is one of the primitive values and the String object is a wrapper around a String primitive.

Strings are written with quotes. You can use single or double quotes.



```
<script>
var carName1 = "Volvo XC60";
var answer1 = "It's alright";
document.getElementById("demo").innerHTML =
carName1 + "<br>" + answer1;
</script>
```

JavaScript Data Types Symbol

A Symbol is a primitive data type whose instances are unique and immutable (unchangeable). In some programming languages they are also called atoms.

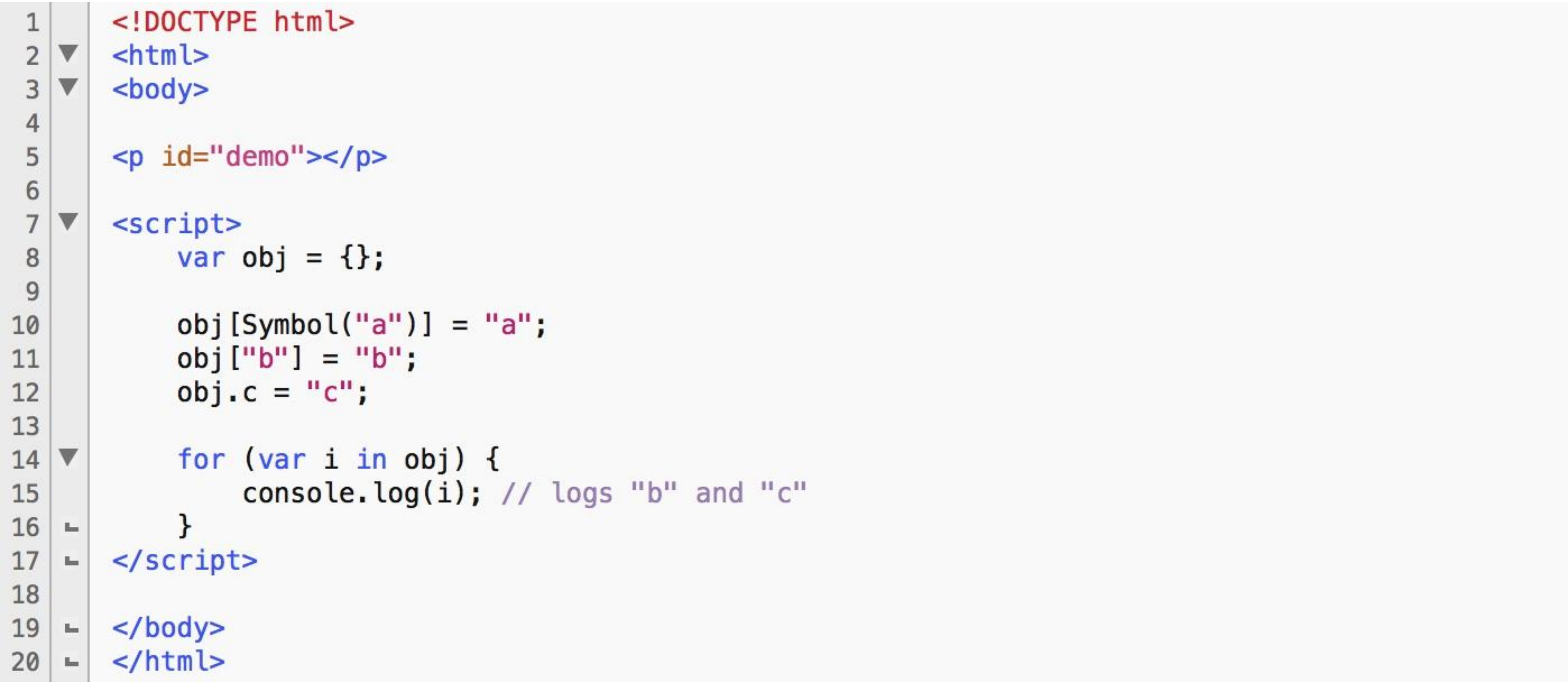
In JavaScript, *Symbol* is one of the primitive values and the Symbol object is a wrapper around a Symbol primitive.

Symbols are not enumerable in `for...in` iterations. In addition, `Object.getOwnPropertyNames()` will not return symbol object properties, however, you can use `Object.getOwnPropertySymbols()` to get these.

Code along

JS Data types - Symbol

Code from the code along



```html
<!DOCTYPE html>
<html>
<body>

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

<script>
    var obj = {};

    obj[Symbol("a")] = "a";
    obj["b"] = "b";
    obj.c = "c";

    for (var i in obj) {
        console.log(i); // logs "b" and "c"
    }
</script>

</body>
</html>
```

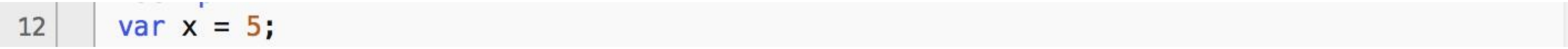
Variables

JavaScript Variables Repetition and detail

We already used JavaScript variables in last lesson, but we'll look a bit more at the concept in detail here.

In JavaScript variables are containers for storing data values.

Creating a variable in JavaScript is called "declaring" a variable.



```
var x = 5;
```

After the declaration, the variable has no value unless set like in the example above. (Technically it has a value of undefined if not set)

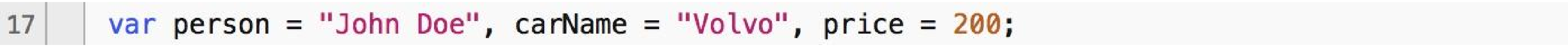
To assign a value to the variable, use the equal sign. In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

JavaScript Variables Repetition and detail

It's a good programming practice to declare all variables at the beginning of a script.

You can declare many variables in one statement.

Start the statement with var and separate the variables by comma:

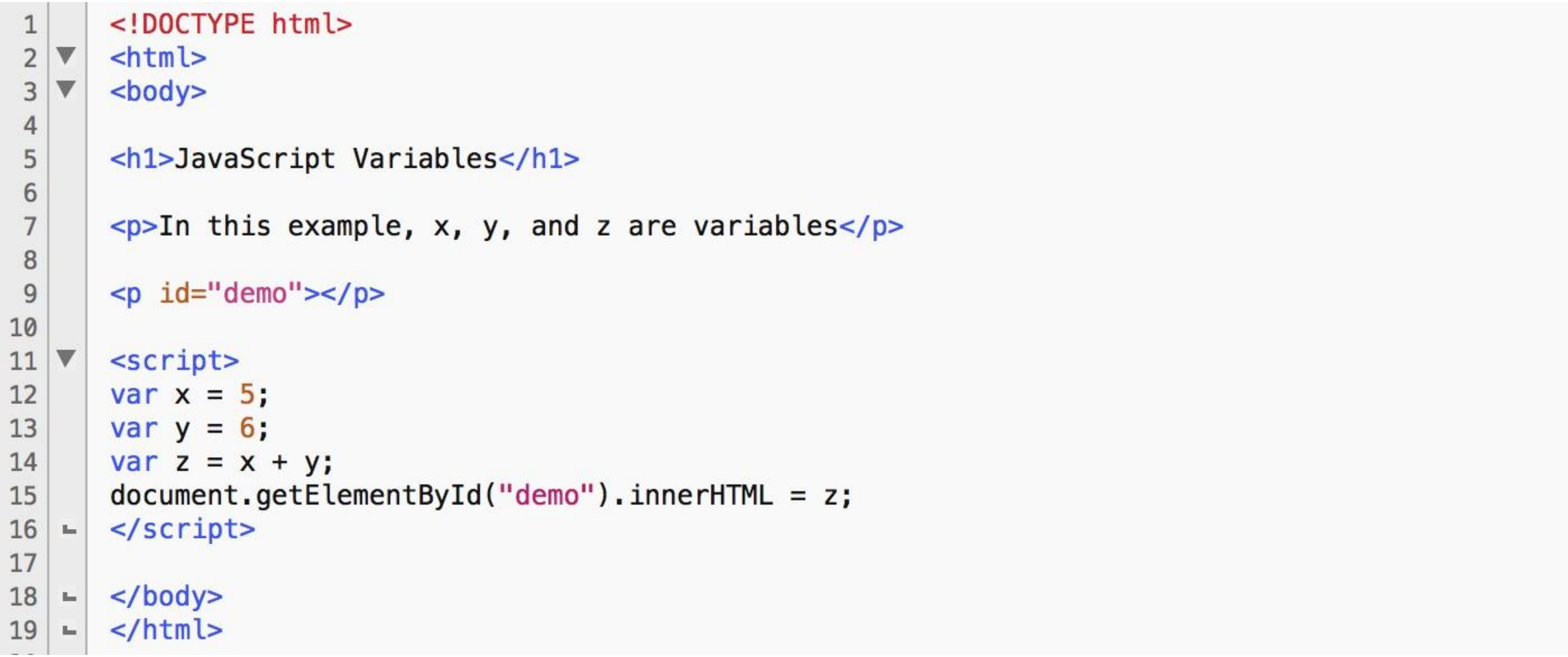


```
var person = "John Doe", carName = "Volvo", price = 200;
```

Code along

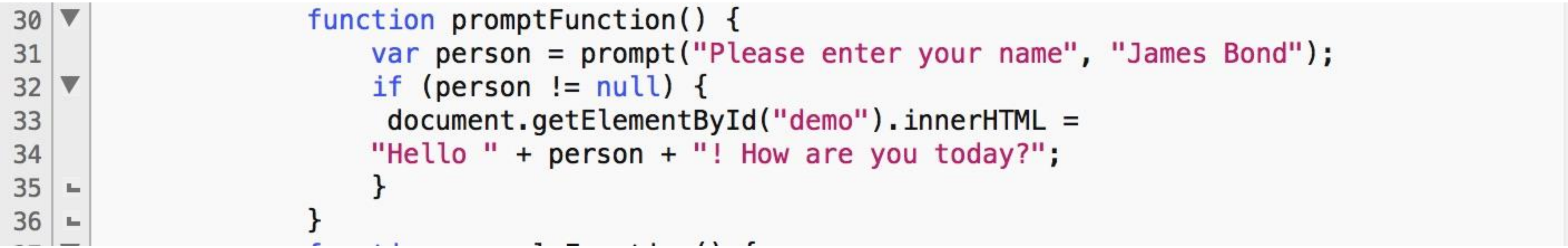
JS Variables
Adding two variables

Code from the code along



```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are variables</p>

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

<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

Example from lesson one where we combined text with variables

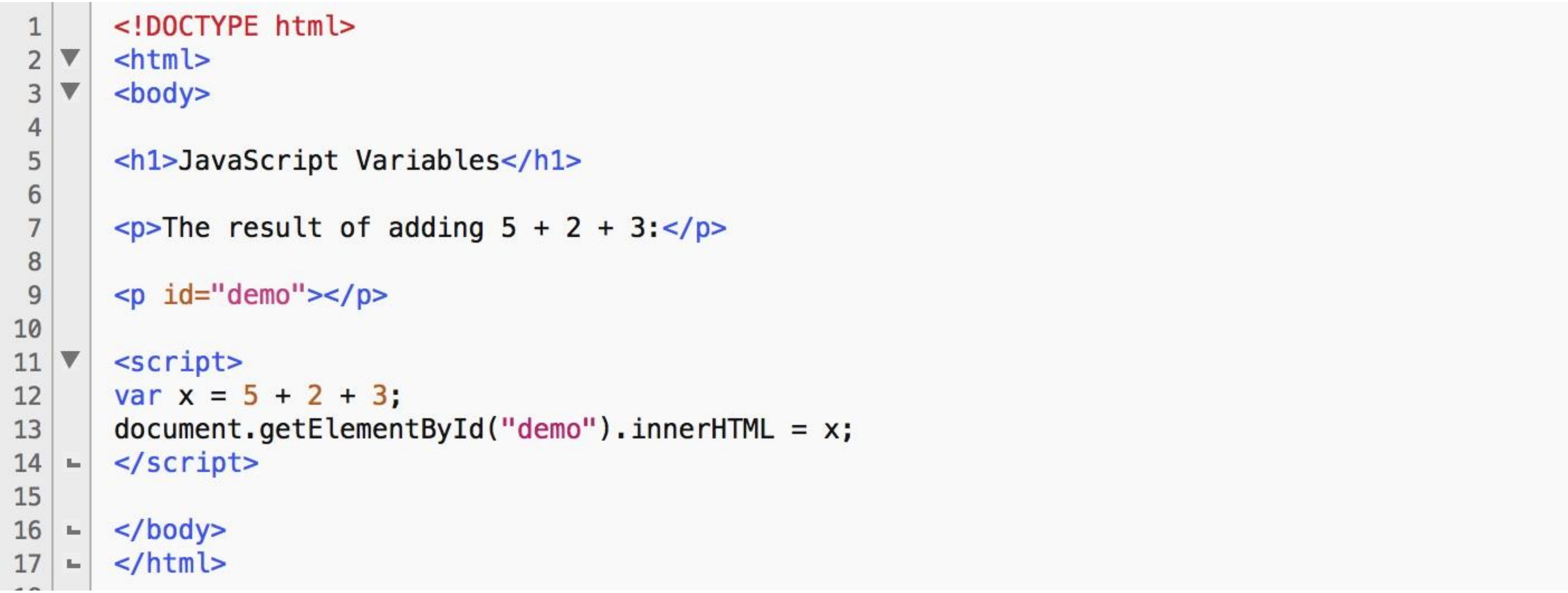


```
function promptFunction() {
    var person = prompt("Please enter your name", "James Bond");
    if (person != null) {
     document.getElementById("demo").innerHTML =
    "Hello " + person + "! How are you today?";
    }
}
```

Code along

JS Variables
Playing with variables

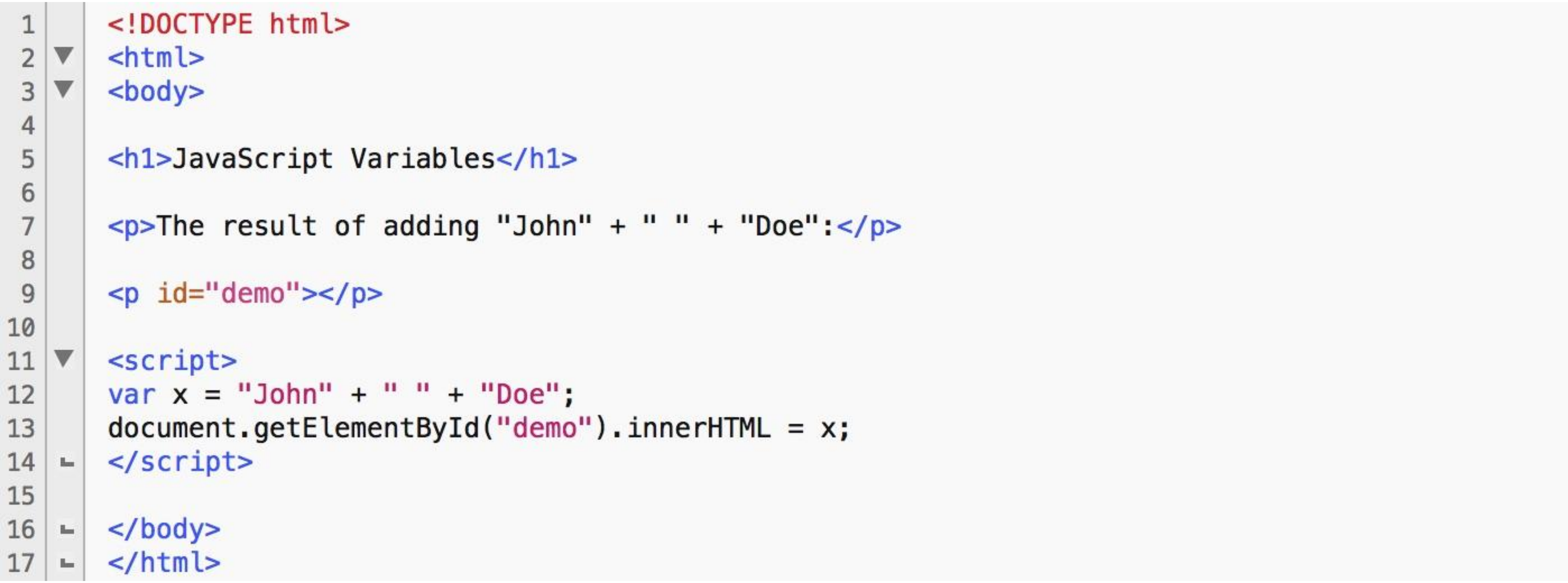
Code from the code along



```html
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>The result of adding 5 + 2 + 3:</p>

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

<script>
var x = 5 + 2 + 3;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +

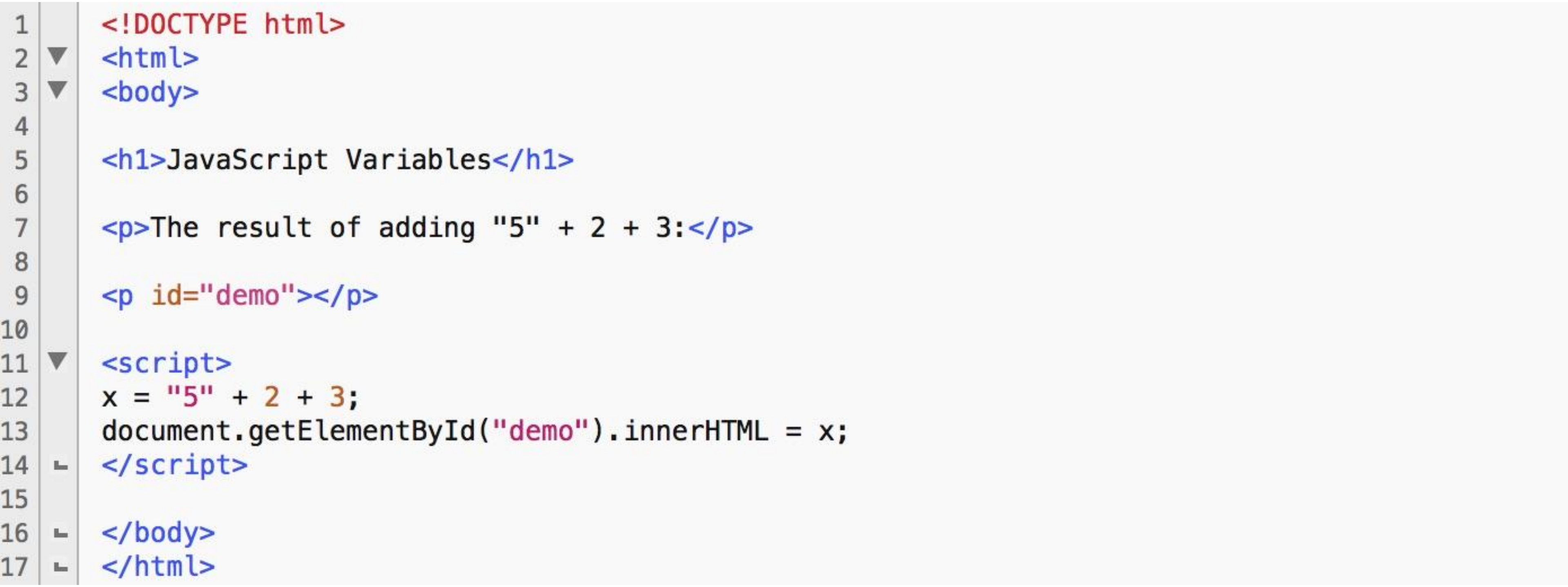
Code from the code along



```html
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>The result of adding "John" + " " + "Doe":</p>

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

<script>
var x = "John" + " " + "Doe";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

You can also add strings, but strings will be concatenated (chained).

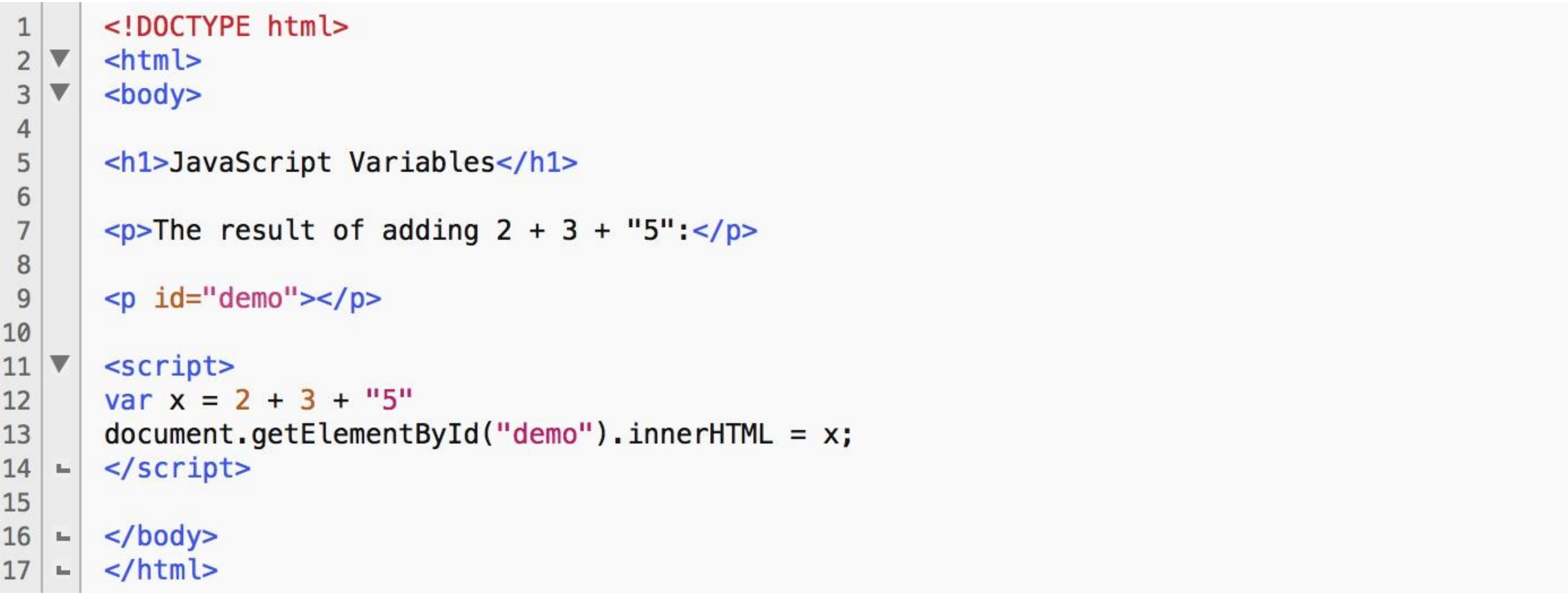
Code from the code along



```html
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>The result of adding "5" + 2 + 3:</p>

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

<script>
x = "5" + 2 + 3;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated...

Code from the code along



```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>The result of adding 2 + 3 + "5":</p>

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

<script>
var x = 2 + 3 + "5"
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

… but if you first put it at the end then only the last number will be considered a string.

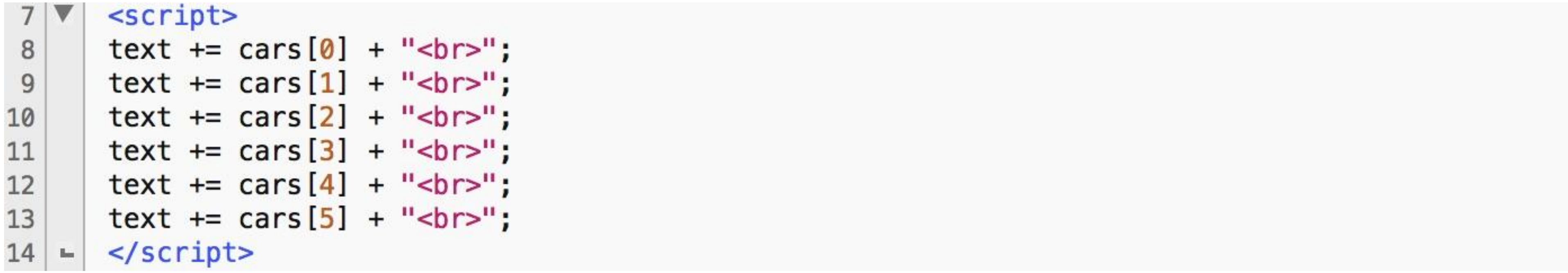
Loops

JavaScript Loops Make the World go round and round and...

We already used JavaScript loops in last lesson, but we’ll look a bit more at the concept in detail here.

Loops are handy, if you want to run the same code over and over again, each time with a different value.

We do not want this:



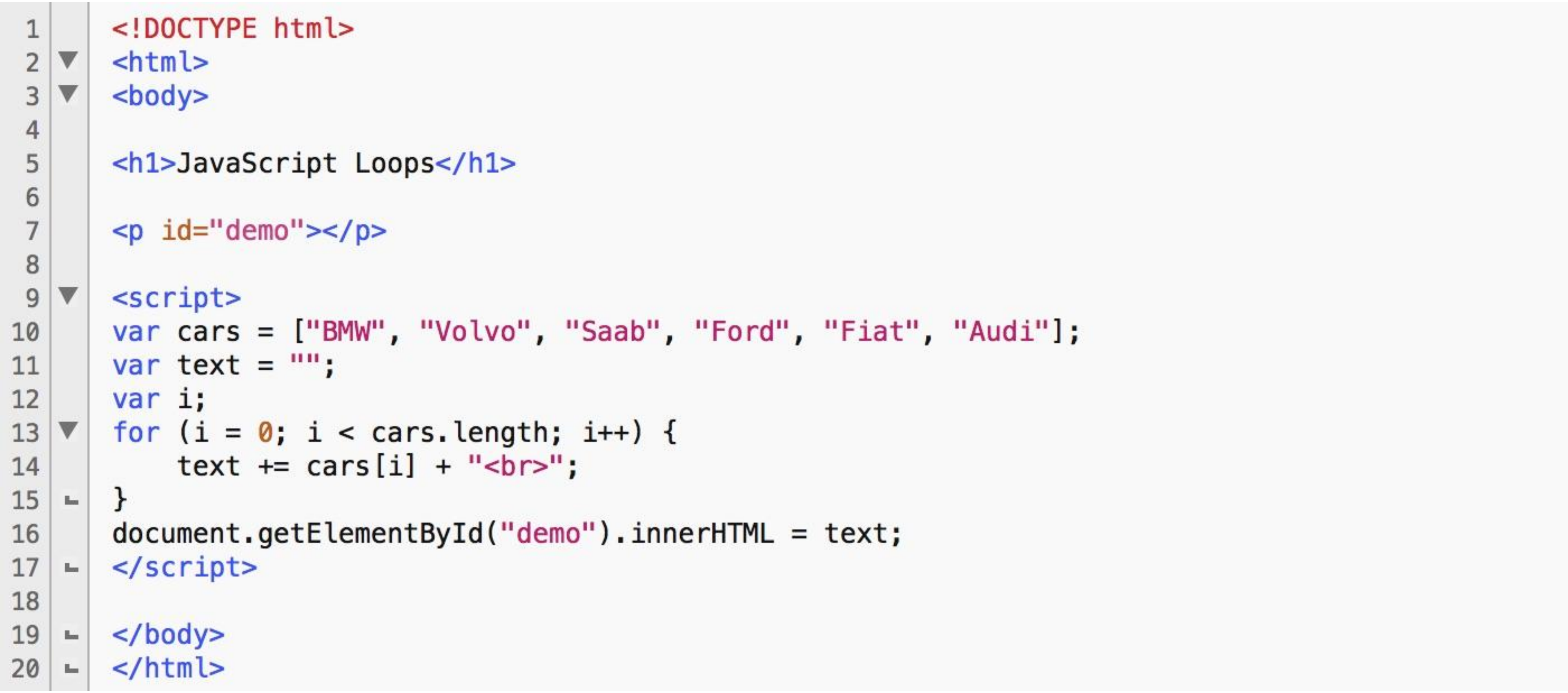
```
<script>
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
</script>
```

So let’s code something better using a loop.

Code along

JS Loops

Code from the code along



```html
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Loops</h1>

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

<script>
var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
var text = "";
var i;
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

i++ increases a value, often used like this where it increases every time it is run (looped)

JavaScript Variables Loops

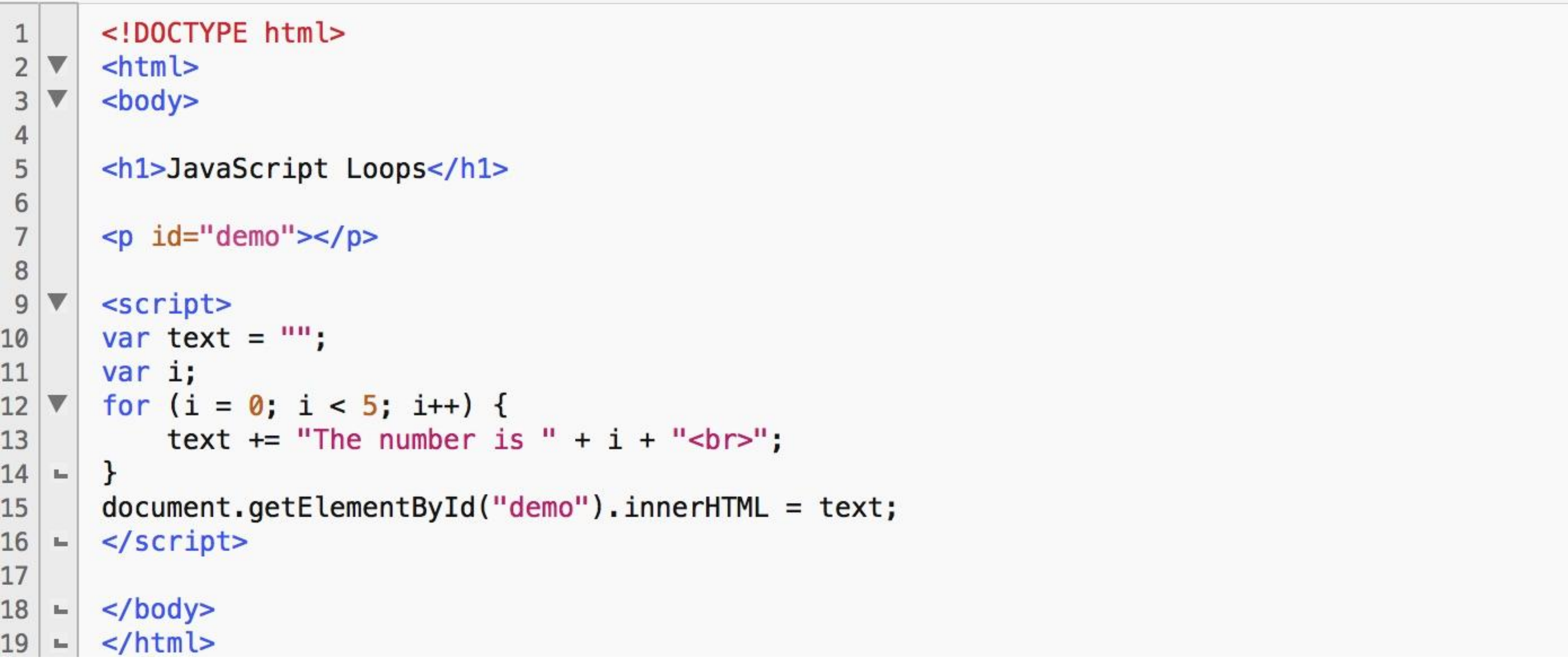
JavaScript supports different kinds of loops:

1. **for** - loops through a block of code a number of times
2. **for/in** - loops through the properties of an object
3. **while** - loops through a block of code while a specified condition is true
4. **do/while** - also loops through a block of code while a specified condition is true

Code along

JS For Loops

Code from the code along



```html
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Loops</h1>

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

<script>
var text = "";
var i;
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Statement 1 (var i = 0;) is executed before the loop (the code block) starts.

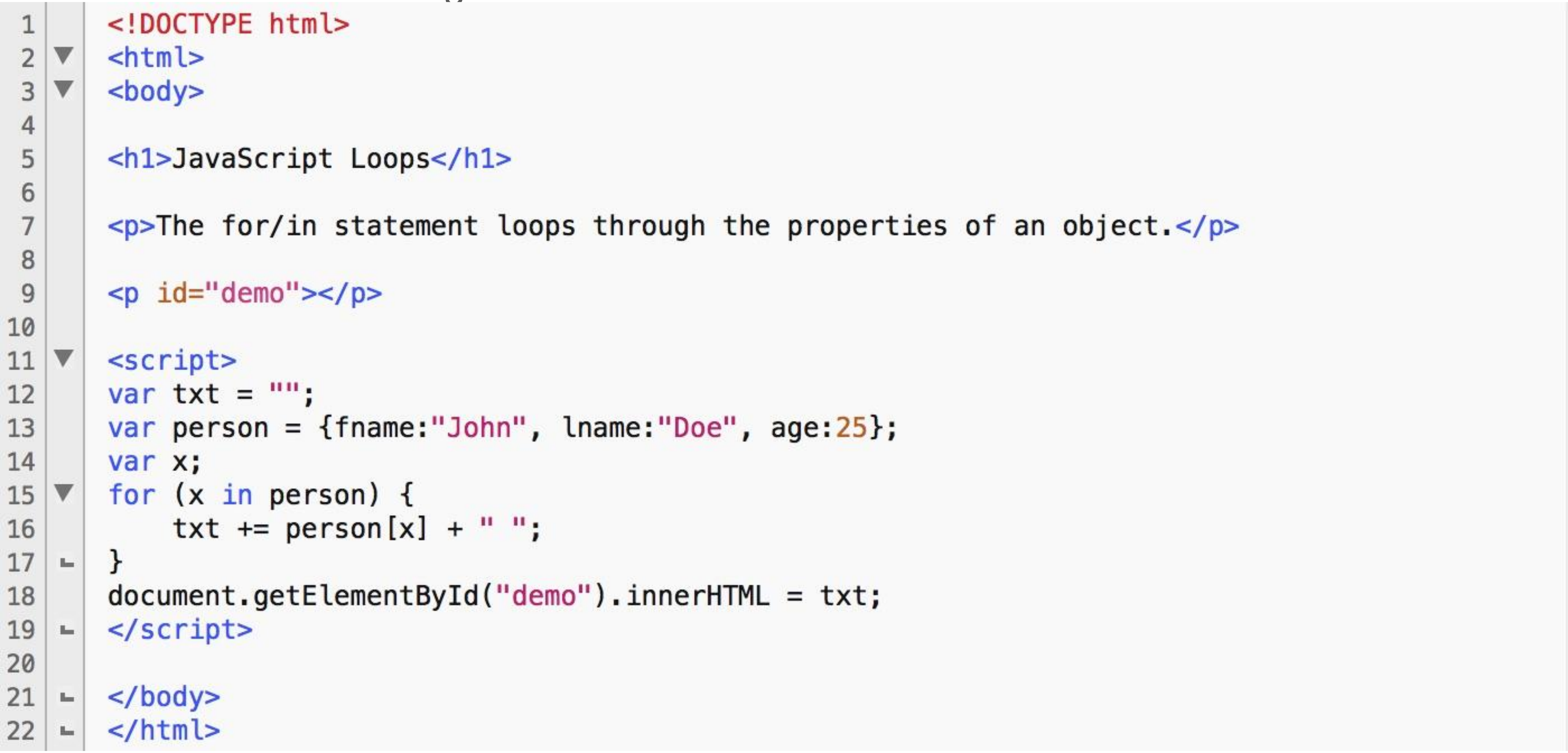
Statement 2 (i < 5;) defines the condition for running the loop (the code block).

Statement 3 (i++) is executed each time after the loop (the code block) has been executed.

Code along

JS For/In Loops

Code from the code along



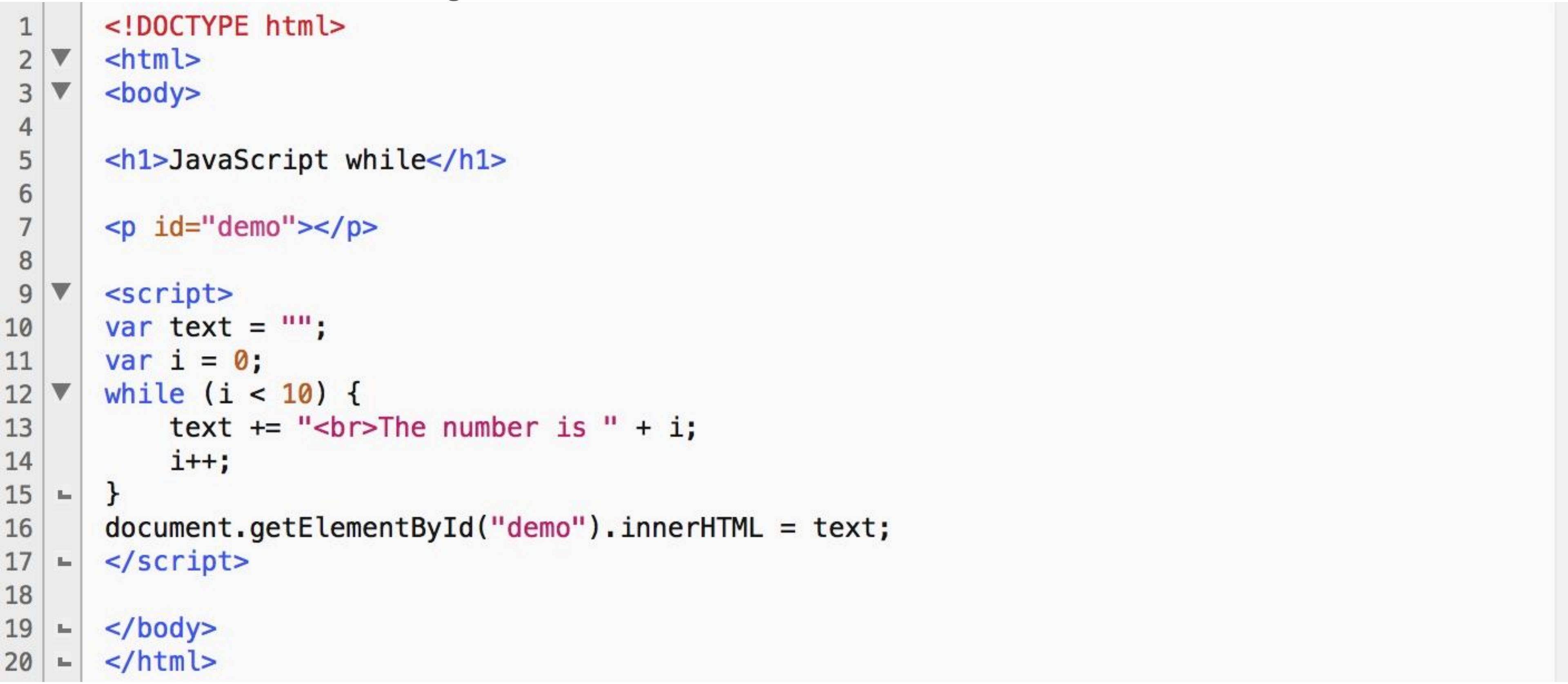
```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Loops</h1>

<p>The for/in statement loops through the properties of an object.</p>

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

<script>
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
    txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

The JavaScript for/in statement loops through the properties of an object.

Code along

JS While Loops

Code from the code along



```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript while</h1>

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

<script>
var text = "";
var i = 0;
while (i < 10) {
    text += "<br>The number is " + i;
    i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

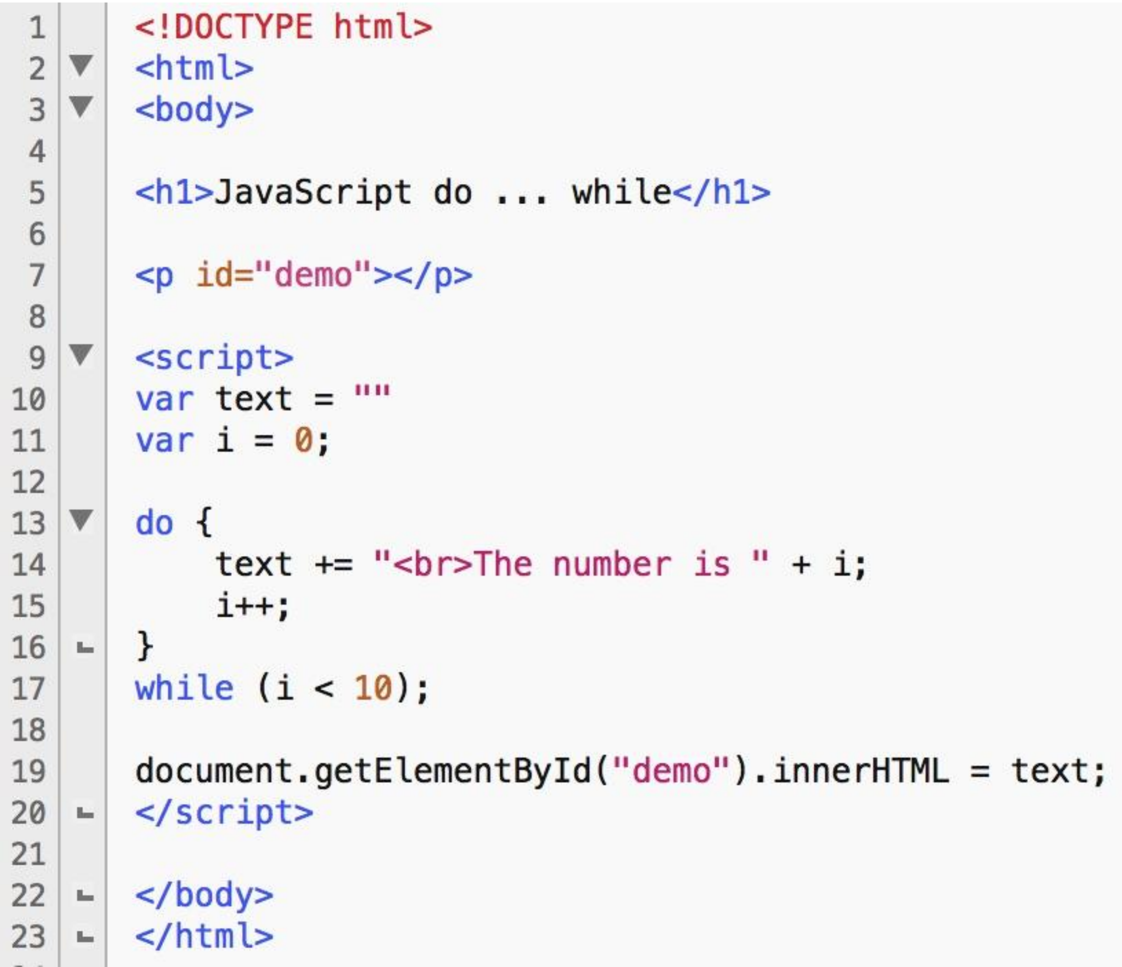
The code in the loop will run, over and over again, as long as a variable (i) is less than 10.

If you forget to increase the variable used in the condition, the loop **will never end**. This will crash your browser.

Code along

JS Do/While Loops

Code from the code along



```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript do ... while</h1>

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

<script>
var text = ""
var i = 0;

do {
     text += "<br>The number is " + i;
     i++;
}
while (i < 10);

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested.

If you forget to increase the variable used in the condition, the loop **will never end**.
This will crash your browser.

Functions

JavaScript Functions Repetition and detail

We already used JavaScript functions in last lesson, but we’ll look a bit more at the concept in detail here.

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

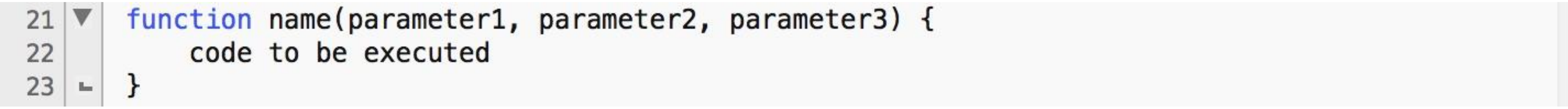
A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(parameter1, parameter2, ...)

JavaScript Functions Repetition and detail

The code to be executed, by the function, is placed inside curly brackets: {}



```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** received by the function when it is invoked.

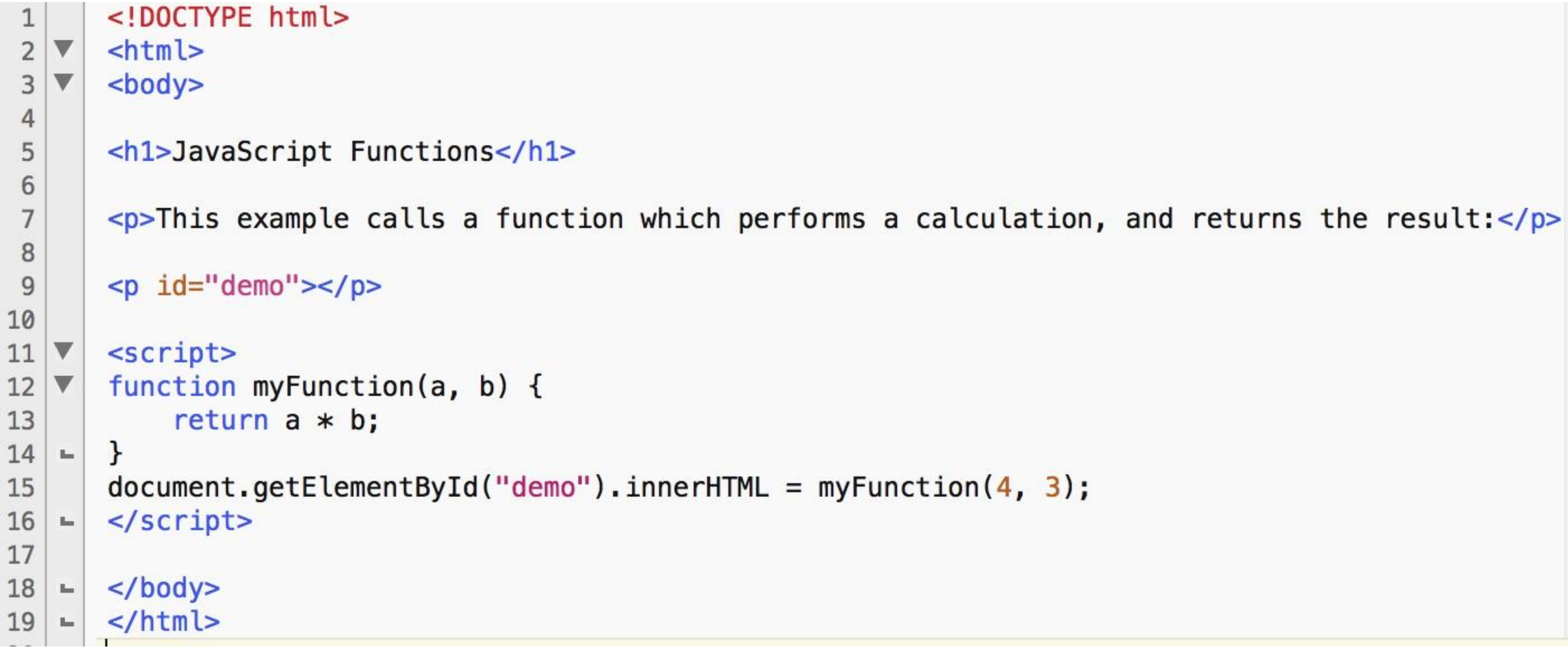
Inside the function, the arguments (the parameters) behave as local variables.

A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Code along

JS Functions

Code from the code along



```html
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Functions</h1>

<p>This example calls a function which performs a calculation, and returns the result:</p>

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

<script>
function myFunction(a, b) {
    return a * b;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>

</body>
</html>
```

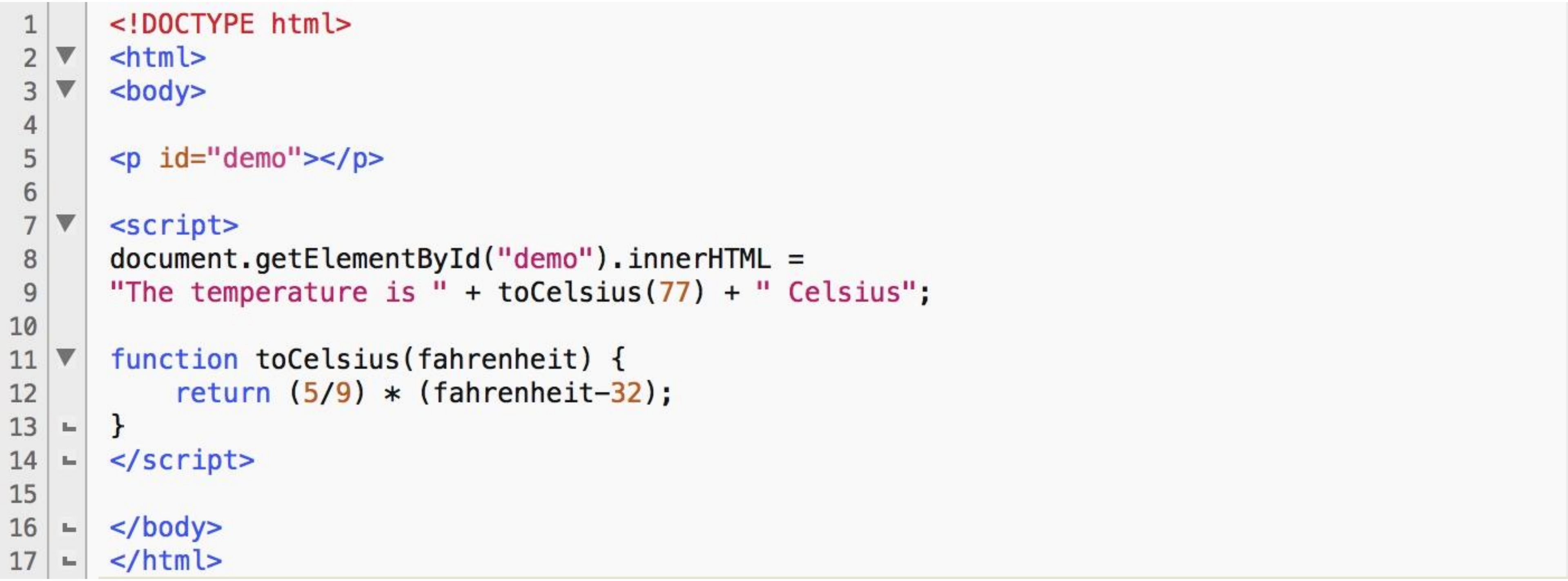
JavaScript Functions Repetition and detail

With functions you can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Code from the code along



```
<!DOCTYPE html>
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML =
"The temperature is " + toCelsius(77) + " Celsius";

function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
</script>

</body>
</html>
```

Instead of using a variable to store the return value of a function, you can use the function directly, as a variable value.

Objects

JavaScript Objects Endless Bags of Holding

You have already learned that JavaScript variables are containers for data values.

Objects are variables too. But objects can contain many values.

JavaScript objects are containers for **named** values.
The values are written as **name**:**value** pairs (name and value separated by a colon).

```
var car = {type:"Fiat", model:"500", color:"white"};
```

The name:values pairs (in JavaScript objects) are called **properties**.

JavaScript Objects Endless Bags of Holding

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
var x = new String();        // Declares x as a String object

var y = new Number();        // Declares y as a Number object

var z = new Boolean();       //  Declares z as a Boolean object
```

It is best to avoid String, Number, and Boolean objects though. They complicate your code and slow down execution speed. Just use them regularly as you can use a primitive as an object just as well in the majority of the cases.

JavaScript Objects Endless Bags of Holding

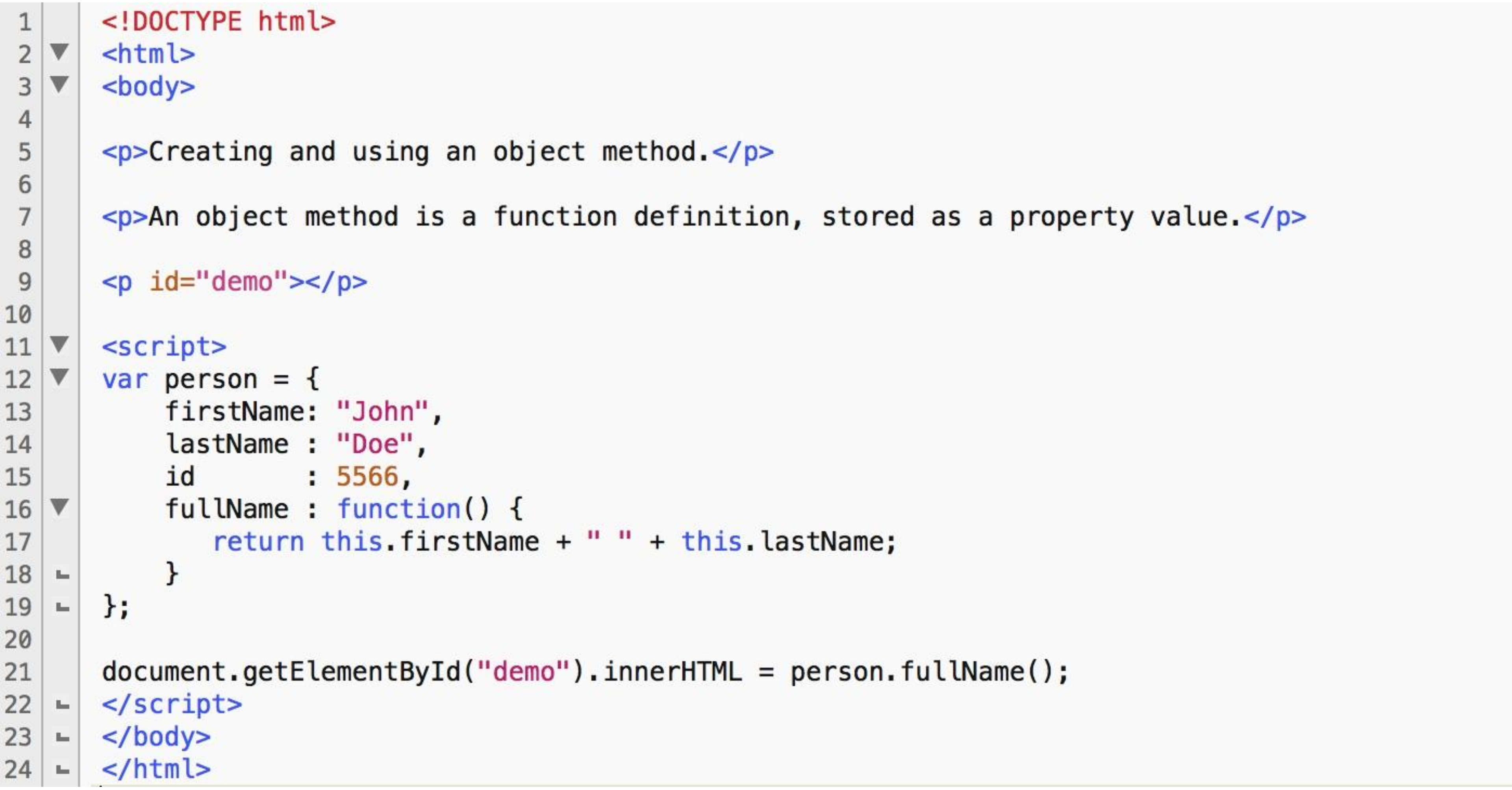
JavaScript objects can not only contain **properties** but also **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Code along

JS Objects



```html
<!DOCTYPE html>
<html>
<body>

<p>Creating and using an object method.</p>

<p>An object method is a function definition, stored as a property value.</p>

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

<script>
var person = {
    firstName: "John",
    lastName : "Doe",
    id       : 5566,
    fullName : function() {
       return this.firstName + " " + this.lastName;
    }
};

document.getElementById("demo").innerHTML = person.fullName();
</script>
</body>
</html>
```

Sources

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
- http://www.w3schools.com/js/js_datatypes.asp
- http://www.w3schools.com/js/js_booleans.asp
- http://www.w3schools.com/js/js_numbers.asp
- http://www.w3schools.com/js/js_loop_for.asp
- http://www.w3schools.com/js/js_loop_while.asp
- http://www.w3schools.com/js/js_functions.asp
- http://www.w3schools.com/js/js_objects.asp