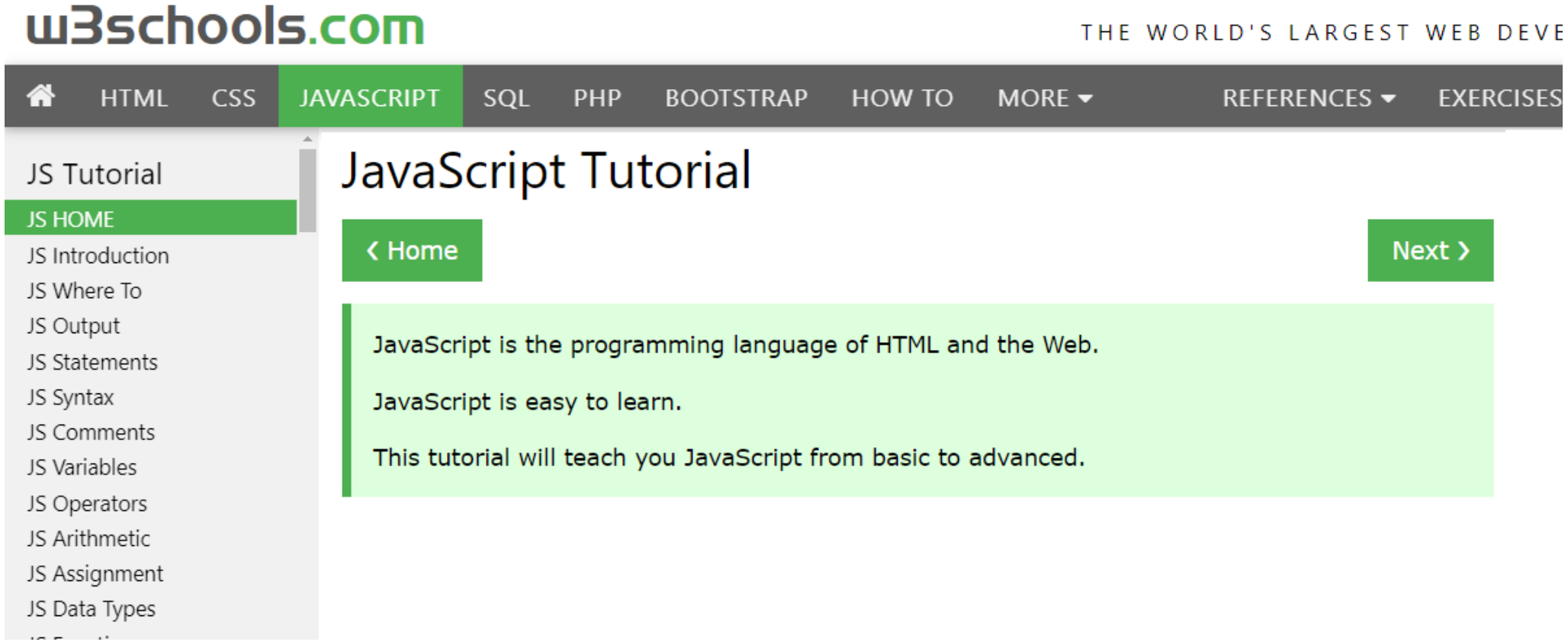
JAVASCRIPT, PART 1

Web Application Development

SOURCE: W3SCHOOLS

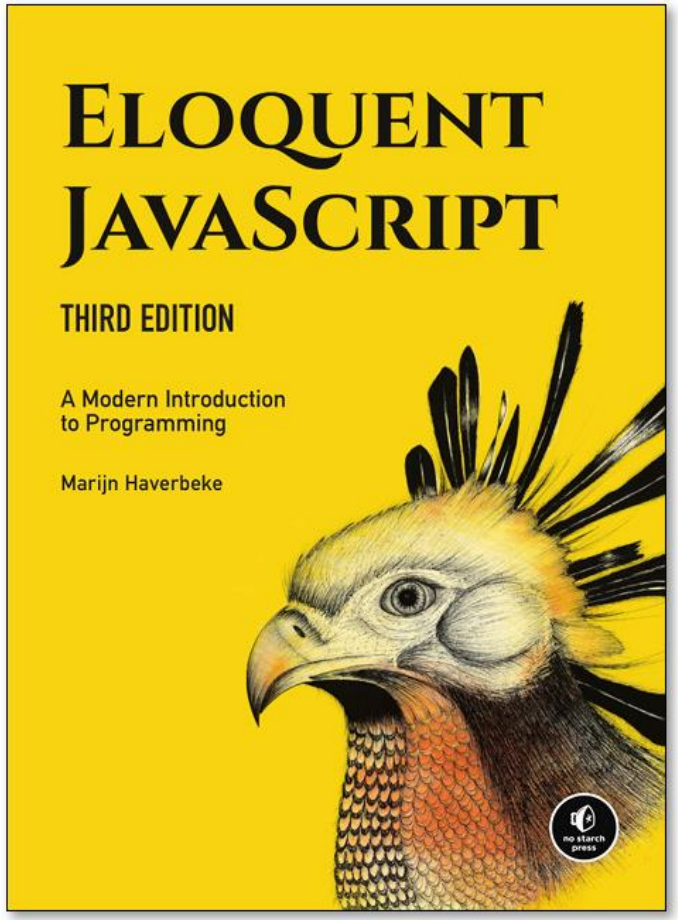


AGENDA

- Introduction
- Where to put JS code
- Output
- Statements
- Comments
- Variables
- Data Types
- Operators
- Arrays
- Functions

Homework: Read the book *Eloquent JavaScript*

ELOQUENT JAVASCRIPT



ELOQUENT JAVASCRIPT
3RD EDITION

CONTENTS

INTRODUCTION

JavaScript

WHAT IS JAVASCRIPT?

- **JavaScript is the programming language of HTML and the Web.**
- **JavaScript is not Java**. JavaScript and Java are completely different languages, both in concept and design.
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997. ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.
- You can read more about the different JavaScript versions here: JS Versions.
- W3Schools maintains a complete JavaScript reference, including all HTML and browser objects. The reference contains examples for all properties, methods and events, and is continuously updated according to the latest web standards. See: Complete JavaScript Reference

WHY STUDY JAVASCRIPT?

- JavaScript is one of the **3 languages** all web developers **must** learn:
- 1. **HTML** to define the content of web pages
- 2. **CSS** to specify the layout of web pages
- 3. **JavaScript** to program the behavior of web pages
- Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

RECALL: DOCUMENT OBJECT MODEL (DOM)

- The browser parses an HTML file into tree structure, called the DOM

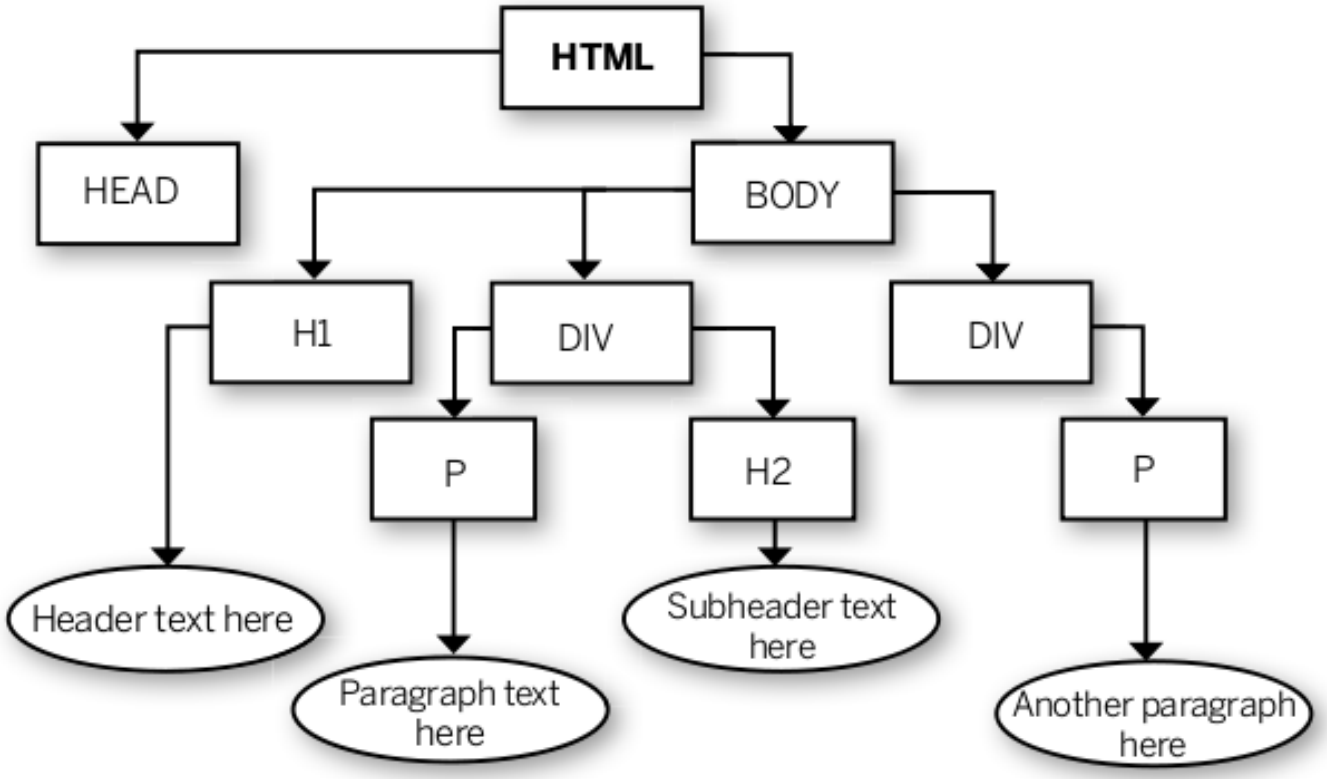


Image source: http://ga-wdi-lessons.github.io/js-dom-jquery-first/dom-tree.png

RECALL: HTML TAGS CAN HAVE ID ATTRIBUTES

```
<img id="myImage" src="pic_bulboff.gif" style="width:100px">
```

JAVASCRIPT INTRODUCTION

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is **getElementById()**. The example below uses the method to "find" an HTML element (with id="demo") and changes the element content (**innerHTML**) to "Hello JavaScript":

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_inner_html

JavaScript accepts both double and single quotes:

```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_inner_html_quotes

EXERCISE 04

- Create a Pen (www.codepen.io) to demonstrate your knowledge of these slides
- Put a link to your pen on your student home page at www.cis255.com

JAVASCRIPT INTRODUCTION

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the src (source) attribute of an <img> tag.

https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_lightbulb


```
<button
onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn
on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button
onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn
off the light</button>
```

JAVASCRIPT INTRODUCTION

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

```
document.getElementById("demo").style.fontSize = "35px";
or
document.getElementById('demo').style.fontSize = '35px';
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_style

JAVASCRIPT INTRODUCTION

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the display style:

```
document.getElementById("demo").style.display = "none";
or
document.getElementById('demo').style.display = 'none';
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_hide

There is a complete list of *.style.display properties at https://www.w3schools.com/jsref/prop_style_display.asp

JAVASCRIPT INTRODUCTION

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the display style:

```
document.getElementById("demo").style.display = "block";
or
document.getElementById('demo').style.display = 'block’;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_show

WHERE TO PUT JS CODE

JAVASCRIPT WHERE TO PUT JS CODE

The <script> Tag

In HTML, **embedded (internal)** JavaScript code must be inserted between <script> and </script> tags, **usually at the bottom of the body section.**

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto

Old JavaScript examples may use a type attribute: <script type="text/JavaScript">. The type attribute is not required. JavaScript is the default scripting language in HTML.

JAVASCRIPT WHERE TO PUT JS CODE

JavaScript Functions and Events

A JavaScript **function** is a block of JavaScript code, that can be executed when "called“ (invoked).

For example, a function can be called when an **event** occurs, like when the user clicks a button.

JAVASCRIPT WHERE TO PUT JS CODE

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JAVASCRIPT WHERE TO PUT JS CODE

JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
<html><head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_head

JAVASCRIPT WHERE TO PUT JS CODE

JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

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

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph
changed.";
}
</script>

</body>
</html>
```

Placing scripts at the bottom of the <body> element improves the display speed, because script compilation slows down the display.

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_body

JAVASCRIPT WHERE TO PUT JS CODE

External JavaScript

Scripts can also be placed in external files:

```
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph
changed.";
}
```

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

JAVASCRIPT WHERE TO PUT JS CODE

External JavaScript Continued

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

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

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_external

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

External scripts cannot contain <script> tags.

JAVASCRIPT WHERE TO PUT JS CODE

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

JAVASCRIPT WHERE TO PUT JS CODE

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_url

This example uses a script located in a specified folder on the current web site:

```
<script src="/js/myScript1.js"></script>
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_url_relative

JAVASCRIPT WHERE TO PUT JS CODE

External References Continued

This example links to a script located in the same folder as the current page:

```
<script src="myScript1.js"></script>
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_external

You can read more about file paths in the chapter HTML File Paths.

JAVASCRIPT WHERE TO PUT JS CODE

Inline event attributes

```
<button onclick="myFunction(); var a = 1;">Click me</button>
```

OUTPUT

JAVASCRIPT OUTPUT

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using **innerHTML**.
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.

JAVASCRIPT OUTPUT

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

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

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

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

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

</body>
</html>
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_dom

JAVASCRIPT OUTPUT

Using document.write()

For testing purposes, it is convenient to use **document.write()**:

```
<h2>My First Web Page</h2>

<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading. It will overwrite the whole document.</p>

<script>

document.write(5 + 6);

</script>
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_write

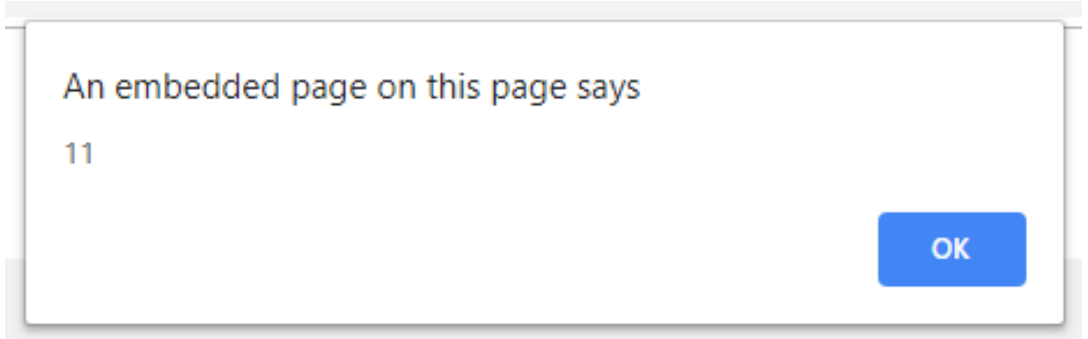
JAVASCRIPT OUTPUT

Using window.alert()

You can use an alert box to display data:

```
<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>
```



Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_alert

JAVASCRIPT OUTPUT

Using console.log()

For debugging purposes, you can use the **console.log()** method to display data.

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

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

You will learn more about debugging in a later chapter.

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_console

STATEMENTS

JavaScript

JAVASCRIPT STATEMENTS

JavaScript Statements

```
var x, y, z;     // Statement 1
x = 5;           // Statement 2
y = 6;           // Statement 3
z = x + y;       // Statement 4
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements

JAVASCRIPT STATEMENTS

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

In HTML, JavaScript programs are executed by the web browser.

JAVASCRIPT STATEMENTS

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_statement

Most JavaScript programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

JAVASCRIPT STATEMENTS

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

```
var a, b, c;      // Declare 3 variables
a = 5;            // Assign the value 5 to a
b = 6;            // Assign the value 6 to b
c = a + b;        // Assign the sum of a and b to c
```

let allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used.

var declares a variable globally, or locally to an entire function regardless of block scope.

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon1

JAVASCRIPT STATEMENTS

Semicolons ; Continued

When separated by semicolons, multiple statements on one line are allowed:

```
a = 5; b = 6; c = a + b;
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon2

On the web, you might see examples without semicolons.
Ending statements with semicolon is not required, but highly recommended.

JAVASCRIPT STATEMENTS

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
var person = "Hege";
var person="Hege";
```

A good practice is to put spaces around operators (= + - * /):

```
var x = y + z;
```

JAVASCRIPT STATEMENTS

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_linebreak

JAVASCRIPT STATEMENTS

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

```
function myFunction() {
    document.getElementById("demo1").innerHTML = "Hello Dolly!";
    document.getElementById("demo2").innerHTML = "How are you?";
}
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_blocks

In this tutorial we use 4 spaces of indentation for code blocks.
You will learn more about functions later in this tutorial.

JAVASCRIPT STATEMENTS

JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

COMMENTS

JAVASCRIPT COMMENTS

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

JAVASCRIPT COMMENTS

Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments1

JAVASCRIPT COMMENTS

Single Line Comments Continued

This example uses a single line comment at the end of each line to explain the code:

```
var x = 5;       // Declare x, give it the value of 5
var y = x + 2;   // Declare y, give it the value of x + 2
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments5

JAVASCRIPT COMMENTS

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments2

It is most common to use single line comments.
Block comments are often used for formal documentation.

JAVASCRIPT COMMENTS

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding // in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments3

JAVASCRIPT COMMENTS

Using Comments to Prevent Execution Continued

This example uses a comment block to prevent execution of multiple lines:

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments4

VARIABLES

JavaScript

JAVASCRIPT VARIABLES

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

```
var x = 5;
var y = 6;
var z = x + y;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables

From the example above, you can expect:

- x stores the value 5
- y stores the value 6
- z stores the value 11

JAVASCRIPT VARIABLES

Much Like Algebra

In this example, price1, price2, and total, are variables:

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_total

In programming, just like in algebra, we use variables (like `price1`) to hold values.

In programming, just like in algebra, we use variables in expressions (`total = price1 + price2`).

From the example above, you can calculate the total to be `11`.

JavaScript variables are containers for storing data values.

JAVASCRIPT VARIABLES

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

JAVASCRIPT VARIABLES

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

The "equal to" operator is written like == in JavaScript.

JAVASCRIPT VARIABLES

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_types

JAVASCRIPT VARIABLES

Declaring (Creating) JavaScript Variables

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

You declare a JavaScript variable with the **var** keyword:

```
var carName;
```

After the declaration, the variable has no value. (Technically it has the value of **undefined**)

To **assign** a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

```
var carName = "Volvo";
```

JAVASCRIPT VARIABLES

Declaring (Creating) JavaScript Variables Continued

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

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


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

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_create

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

JAVASCRIPT VARIABLES

One Statement, Many Variables

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;
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multi

A declaration can span multiple lines:

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

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multiline

JAVASCRIPT VARIABLES

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value **undefined**.

The variable carName will have the value undefined after the execution of this statement:

```
var carName;
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_undefined

JAVASCRIPT VARIABLES

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

```
var carName = "Volvo";
var carName;
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_redefine

JAVASCRIPT VARIABLES

JavaScript Arithmetic

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

```
var x = 5 + 2 + 3;
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_numbers

You can also add strings, but strings will be concatenated:

```
var x = "John" + " " + "Doe";
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_strings

JAVASCRIPT VARIABLES

JavaScript Arithmetic Continued

Also try this:

```
var x = "5" + 2 + 3;
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_string_number

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

Now try this:

```
var x = 2 + 3 + "5";
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_number_string

DATA TYPES

JavaScript

JAVASCRIPT DATA TYPES

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var x = {firstName:"John", lastName:"Doe"};         // Object
```

JAVASCRIPT DATA TYPES

Primitive data types

- Boolean (true or false)
- Null
- Undefined
- Number (not int, float, double, etc.): can be any real number or +Infinity, -Infinity, and NaN (not-a-number).
- String
- Symbol (new in ECMAScript 6). A Symbol is a **unique** and **immutable** primitive value and may be used as the key of an Object property. This is an advanced topic not used in this course.
- Object (a collection of properties)

JAVASCRIPT DATA TYPES

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
var x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
var x = "16" + "Volvo";
```

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

JAVASCRIPT DATA TYPES

The Concept of Data Types Continued

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

JavaScript:

```
var x = 16 + 4 + "Volvo";
```

Result:

```
20Volvo
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_1

JAVASCRIPT DATA TYPES

The Concept of Data Types Continued

JavaScript:

```
var x = "Volvo" + 16 + 4;
```

Result:

```
Volvo164
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_2

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

JAVASCRIPT DATA TYPES

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

```
var x;               // Now x is undefined
x = 5;               // Now x is a Number
x = "John";          // Now x is a String
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_dynamic

JAVASCRIPT DATA TYPES

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

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

```
var carName = "Volvo XC60";   // Using double quotes
var carName = 'Volvo XC60';   // Using single quotes
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string_quotes

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
var answer = "It's alright";             // Single quote inside double quotes
var answer = "He is called 'Johnny'";    // Single quotes inside double quotes
var answer = 'He is called "Johnny"';    // Double quotes inside single quotes
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string

You will learn more about strings later in this tutorial.

JAVASCRIPT DATA TYPES

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

```
var x1 = 34.00;      // Written with decimals
var x2 = 34;         // Written without decimals
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers

JAVASCRIPT DATA TYPES

JavaScript Numbers Continued

Extra large or extra small numbers can be written with scientific (exponential) notation:

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers_large

You will learn more about numbers later in this tutorial.

JAVASCRIPT DATA TYPES

JavaScript Booleans

Booleans can only have two values: true or false.

```
var x = 5;
var y = 5;
var z = 6;
(x == y)        // Returns true
(x == z)        // Returns false
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_bolean

Booleans are often used in conditional testing.

You will learn more about conditional testing later in this tutorial.

JAVASCRIPT DATA TYPES

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

```
var cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_array

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about arrays later in this tutorial.

JAVASCRIPT DATA TYPES

JavaScript Objects

JavaScript objects are written with curly braces.

Object properties are written as name:value pairs, separated by commas.

```
var person = {
      firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"
};
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_object

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

You will learn more about objects later in this tutorial.

JAVASCRIPT DATA TYPES

The typeof Operator

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable.

The **typeof** operator returns the type of a variable or an expression:

```
typeof ""                    // Returns "string"
typeof "John"                // Returns "string"
typeof "John Doe"            // Returns "string"
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_string

```
typeof 0                     // Returns "number"
typeof 314                   // Returns "number"
typeof 3.14                  // Returns "number"
typeof (3)                   // Returns "number"
typeof (3 + 4)               // Returns "number"
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_number

JAVASCRIPT DATA TYPES

Undefined

In JavaScript, a variable without a value, has the value **undefined**. The typeof is also **undefined**.

```
var car;                    // Value is undefined, type is undefined
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined

Any variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

```
car = undefined;            // Value is undefined, type is undefined
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_2

JAVASCRIPT DATA TYPES

Empty Values

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

```
var car = "";    // The value is "", the typeof is "string"
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_empty

JAVASCRIPT DATA TYPES

Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

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

You can consider it a bug in JavaScript that typeof null is an object. It should be null.

You can empty an object by setting it to null:

```
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
person = null; // Now value is null, but type is still an object
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_null

JAVASCRIPT DATA TYPES

Null Continued

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

```
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
person = undefined;   // Now both value and type is undefined
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_1

JAVASCRIPT DATA TYPES

Difference Between Undefined and Null

Undefined and null are equal in value but different in type:

```
typeof undefined               // undefined
typeof null                    // object

null === undefined             // false
null == undefined              // true
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_3

JAVASCRIPT DATA TYPES

Primitive Data

A primitive data value is a single simple data value with no additional properties and methods.

The **typeof** operator can return one of these primitive types:

- String, number, Boolean, undefined, object

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof true                   // Returns "boolean"
typeof false                  // Returns "boolean"
typeof x                      // Returns "undefined" (if x has no
value)
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_primitive

JAVASCRIPT DATA TYPES

Complex Data

The **typeof** operator can return one of two complex types:

- function
- object

The typeof operator returns **object** for objects, arrays, and null.

The typeof operator does not return object for functions.

```
typeof {name:'John', age:34} // Returns "object"
typeof [1,2,3,4]             // Returns "object" (not "array", see note
below)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_complex

The typeof operator returns "object" for arrays because in JavaScript arrays are objects.

OPERATORS

JavaScript

JAVASCRIPT OPERATORS

Assign values to variables and add them together:

```
var x = 5;          // assign the value 5 to x
var y = 2;          // assign the value 2 to y
var z = x + y;      // assign the value 7 to z (x + y)
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper

The **assignment** operator (=) assigns a value to a variable.

```
var x = 10;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_equal

JAVASCRIPT OPERATORS

The **addition** operator (+) adds numbers:

```
var x = 5;
var y = 2;
var z = x + y;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_add

The **multiplication** operator (*) multiplies numbers.

```
var x = 5;
var y = 2;
var z = x * y;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mult

JAVASCRIPT OPERATORS

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Arithmetic operators are fully described in the JS Arithmetic chapter.

JAVASCRIPT OPERATORS

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

JAVASCRIPT OPERATORS

JavaScript Assignment Operators Continued

The **addition assignment** operator (+=) adds a value to a variable.

```
var x = 10;
x += 5;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_plusequal

Assignment operators are fully described in the JS Assignment chapter.

JAVASCRIPT OPERATORS

JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

```
var txt1 = "John";
var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
```

The result of txt3 will be:

```
John Doe
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concatenate

JAVASCRIPT OPERATORS

JavaScript String Operators Continued

The += assignment operator can also be used to add (concatenate) strings:

```
var txt1 = "What a very ";
txt1 += "nice day";
```

The result of txt1 will be:

```
What a very nice day
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat4

When used on strings, the + operator is called the concatenation operator.

JAVASCRIPT OPERATORS

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
```

If you add a number and a string, the result will be a string!

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat5

JAVASCRIPT OPERATORS

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Comparison operators are fully described in the JS Comparisons chapter.

JAVASCRIPT OPERATORS

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

Logical operators are fully described in the JS Comparisons chapter.

JAVASCRIPT OPERATORS

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Type operators are fully described in the JS Type Conversion chapter.

ARRAYS

JavaScript

JAVASCRIPT ARRAYS

JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.

```
var cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array

JAVASCRIPT ARRAYS

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

JAVASCRIPT ARRAYS

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

var *array_name* = [*item1*, *item2*, ...];

Example:

```
var cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array

JAVASCRIPT ARRAYS

Creating an Array Continued

Spaces and line breaks are not important. A declaration can span multiple lines:

```
var cars = [
    "Saab",
    "Volvo",
    "BMW"
];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_newlines

Putting a comma after the last element (like "BMW",) is inconsistent across browsers. IE 8 and earlier will fail.

JAVASCRIPT ARRAYS

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new

The two examples above do exactly the same. There is no need to use new Array().
For simplicity, readability and execution speed, use the first one (the array literal method).

JAVASCRIPT ARRAYS

Access the Elements of an Array

You access an array element by referring to the **index number.**

This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

Example:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_element

Array indexes start with 0.
[0] is the first element. [1] is the second element.

JAVASCRIPT ARRAYS

Changing an Array Element

This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

Example:

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_element_change

JAVASCRIPT ARRAYS

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_full

JAVASCRIPT ARRAYS

Arrays are Objects

Arrays are a special type of objects. The **typeof** operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, **person[0]** returns John:

```
var person = ["John", "Doe", 46];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_array

Objects use **names** to access its "members". In this example, **person.firstName** returns John:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_object

JAVASCRIPT ARRAYS

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

JAVASCRIPT ARRAYS

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

```
var x = cars.length;   // The length property returns the number of
elements
var y = cars.sort();   // The sort() method sorts arrays
```

Array methods are covered in the next chapters.

JAVASCRIPT ARRAYS

The length Property

The **length** property of an array returns the length of an array (the number of array elements).

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;                          // the length of fruits is 4
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_length

The length property is always one more than the highest array index.

JAVASCRIPT ARRAYS

Accessing the First Array Element

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_first

JAVASCRIPT ARRAYS

Accessing the Last Array Element

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length - 1];
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_last

JAVASCRIPT ARRAYS

Looping Array Elements

The safest way to loop through an array, is using a "for" loop:

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
    text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_loop

JAVASCRIPT ARRAYS

Looping Array Elements Continued

You can also use the Array.forEach() function:

```
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
    text += "<li>" + value + "</li>";
}
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_loop_foreach

JAVASCRIPT ARRAYS

Adding Array Elements

The easiest way to add a new element to an array is using the push method:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");                    // adds a new element (Lemon)
to fruits
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add_push

New element can also be added to an array using the length property:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";     // adds a new element (Lemon)
to fruits
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add

JAVASCRIPT ARRAYS

Adding Array Elements Continued

WARNING !

Adding elements with high indexes can create undefined "holes" in an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";                    // adds a new element (Lemon)
to fruits
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_holes

JAVASCRIPT ARRAYS

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;          // person.length will return 3
var y = person[0];              // person[0] will return "John"
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_1

JAVASCRIPT ARRAYS

Associative Arrays Continued

WARNING !!
If you use named indexes, JavaScript will redefine the array to a standard object.
After that, some array methods and properties will produce **incorrect results**.

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length;          // person.length will return 0
var y = person[0];              // person[0] will return undefined
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_2

JAVASCRIPT ARRAYS

The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes.**

In JavaScript, **objects** use **named indexes.**

Arrays are a special kind of objects, with numbered indexes.

JAVASCRIPT ARRAYS

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

JAVASCRIPT ARRAYS

Avoid new Array()

There is no need to use the JavaScript's built-in array constructor **new** Array().

Use [] instead.

These two different statements both create a new empty array named points:

```
var points = new Array();          // Bad
var points = [];                   // Good
```

These two different statements both create a new array containing 6 numbers:

```
var points = new Array(40, 100, 1, 5, 25, 10); // Bad
var points = [40, 100, 1, 5, 25, 10];          // Good
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_literal

JAVASCRIPT ARRAYS

Avoid new Array() Continued

The **new** keyword only complicates the code. It can also produce some unexpected results:

```
var points = new Array(40, 100);  // Creates an array with two
elements (40 and 100)
```

What if I remove one of the elements?

```
var points = new Array(40);         // Creates an array with 40
undefined elements !!!!!
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new_error

JAVASCRIPT ARRAYS

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator **typeof** returns "object"

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

typeof fruits;                  // returns object
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_typeof

The typeof operator returns object because a JavaScript array is an object.

JAVASCRIPT ARRAYS

How to Recognize an Array – Solution 1

To solve this problem ECMAScript 5 defines a new method **Array.isArray()**:

```
Array.isArray(fruits);     // returns true
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray_method

The problem with this solution is that ECMAScript 5 is **not supported in older browsers**.

JAVASCRIPT ARRAYS

How to Recognize an Array – Solution 2

To solve this problem you can create your own isArray() function:

```
function isArray(x) {
    return x.constructor.toString().indexOf("Array") > -1;
}
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray

The function above always returns true if the argument is an array.

Or more precisely: it returns true if the object prototype contains the word "Array".

JAVASCRIPT ARRAYS

How to Recognize an Array – Solution 3

The **instanceof** operator returns true if an object is created by a given constructor:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits instanceof Array     // returns true
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_instanceof

JAVASCRIPT ARRAYS

Test Yourself with Exercises!

- Exercise 1
- Exercise 2
- Exercise 3

FUNCTIONS

JavaScript

JAVASCRIPT FUNCTIONS

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).

```
function myFunction(p1, p2) {
    return p1 * p2;                 // The function returns the product
of p1 and p2
}
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_functions

JAVASCRIPT FUNCTIONS

JavaScript Function Syntax

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, ...*)

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

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

JAVASCRIPT FUNCTIONS

JavaScript Function Syntax Continued

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **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.

JAVASCRIPT FUNCTIONS

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

JAVASCRIPT FUNCTIONS

Function Return

When JavaScript reaches a **return statement**, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);     // Function is called, return value will end up in
x

function myFunction(a, b) {
    return a * b;             // Function returns the product of a and b
}
```

The result in X will be:

```
12
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_return

JAVASCRIPT FUNCTIONS

Why 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.

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius

JAVASCRIPT FUNCTIONS

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Accessing a function without () will return the function definition instead of the function result:

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Try it Yourself:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius_2

JAVASCRIPT FUNCTIONS

Functions Used as Variable Values

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

Instead of using a variable to store the return value of a function:

```
var x = toCelsius(77);
var text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_variable

You will learn a lot more about functions later in this tutorial.

JAVASCRIPT FUNCTIONS

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

```
// code here can NOT use carName

function myFunction() {
    var carName = "Volvo";
    // code here CAN use carName
}

// code here can NOT use carName
```

Try it Yourself: https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_scope

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

EXERCISE

Web Application Development

EXERCISE 05

- Refer to code on following slide
- See: https://www.w3schools.com/jsref/met_node_appendchild.asp
- Create a web page where user enters a comma separated list in a textarea, then clicks a button to generate the list as an unordered list.
- Put a link on your student home page

EXERCISE

```
<ul id="myList">
  <li>Coffee</li>
  <li>Tea</li>
</ul>

<textarea id="ta" rows="4" cols="50"></textarea>

<p>Click the button to append an item to the end of the list.</p>

<button onclick="myFunction()">Try it</button>
```

```
<script>
function myFunction() {
    var node =
        document.createElement("LI");
    var tx =
        document.getElementById("ta").value;
    var textnode = document.createTextNode(tx);
    node.appendChild(textnode);

        document.getElementById("myList").appendChild(node);
}
</script>
```

THE END