Programming Web

Dr. Jamil Alagha
College of Engineering and Artificial Intelligence
University of Palestine

JavaScript

Agenda

1. Introduction
2. JS Output
3. JS Statements
4. JS Syntax
5. JS Comments
6. JS Variables
7. JS Assignments
8. JS Data Types
9. JS Functions
10. JS Math Objects
11. JS Conditions
12. JS Loops
13. JS Events
14. JS Forms

Introduction

What is JavaScript?

- JavaScript is the programming language of HTML and the Web.
- JavaScript was introduced in 1995 as a way to add programs to web pages in the Netscape Navigator browser.
- The language has since been adopted by all other major graphical web browsers.
- It has made modern web applications possible. Applications with which you can interact directly without doing a page reload for every action.

Why Study JavaScript?

- JavaScript is one of the 3 languages all web developers must learn:
  - **HTML** to define the content of web pages
  - **CSS** to specify the layout of web pages
  - **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.

What Can JavaScript Do?

- These are some examples of what JS can do with HTML:
  - JS can change HTML content
  - JS can change HTML attribute values
  - JS can change HTML styles (CSS)
  - JS can hide HTML elements
  - JS can show HTML elements
  - REACT TO EVENTS (A button is pressed)

- For example:
  - Add a paragraph of text after the first <hl> element
  - Change the value of class attributes to trigger new CSS rules for those elements
  - Change the size or position of an <img> element

Where To Write JS?

- **Internal JS**

  The **<script>** Tag: In HTML, JavaScript code must be inserted between <script> and </script> tags.

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

```
<script>
  … JavaScript code goes here
</script>
```

- **External JS**

  Scripts can also be placed in external ***.js** files. To use an external script, put the name of the script file in the **src** attribute of a **<script>** tag

<script **src=**"my_script.js"></script>

Where To Write JS?

- **Internal JS**

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

- **External JS**

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

JS Output

JavaScript Output

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using **innerHTML**.
    - Element is accessed using *document.getElementById()* function
  - Writing into the HTML output using **document.write()**.
  - Writing into an alert box, using **window.alert()**.
  - Writing into the browser console, using **console.log()**.

Using innerHTML

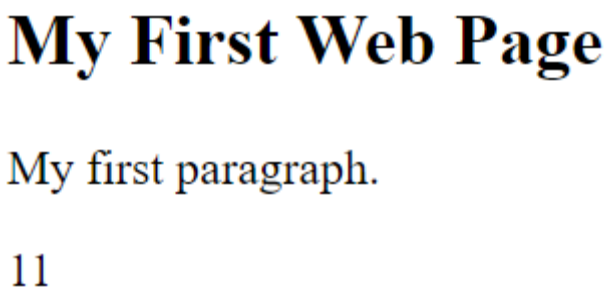
- To access an HTML element, JavaScript can use the **document.getElementById(id)** method.
- The **id** attribute defines the HTML element.
- The **innerHTML** property defines the HTML content:

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

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

Using document.write()

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

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

<script>
document.write(5 + 6);
</script>
```



- Using document.write() after an HTML document is loaded, will **delete all** existing HTML.

Using window.alert()

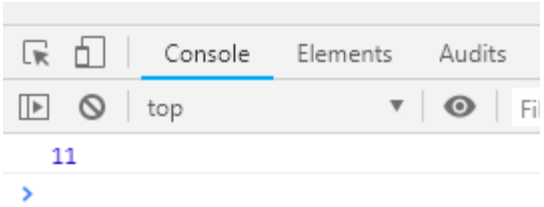
- You can use an alert box to display data:

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

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

Using console.log()

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

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



JS input

Js input

In JavaScript, we can get user input like this:

```
var name = window.prompt("Enter your name: ");
alert("Your name is " + name);
```

JS 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 are composed of:
  - Values
  - Operators
  - Expressions
  - Keywords
  - Comments
- Most JavaScript programs contain many JavaScript statements.
- The statements are executed, one by one, in the same order as they are written.
- Semicolons ; separate JavaScript statements.
- When separated by semicolons, multiple statements on one line are allowed.

JavaScript Statements

```
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


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

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";
```

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!";
```

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?";
}
```

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

JS Syntax

JavaScript Values

- The JavaScript syntax defines two types of values: Fixed values and variable values.
- Fixed values are called **literals**.
  - Numbers are written with or without decimals
  - Strings are text, written within double or single quotes
- Variable values are called **variables**.
  - Variables are used to store data values.
  - JavaScript uses the **var** keyword to declare variables
  - An **equal** sign is used to assign values to variables

```
var x;

x = 6;
```

JavaScript Operators

- JavaScript uses arithmetic operators (**+ - * /**) to compute values.

```
(5 + 6) * 10
```

- JavaScript uses an assignment operator (**=**) to assign values to variables

```
var x, y;
x = 5;
y = 6;
```

JavaScript Expressions

- An **expression** is a combination of values, variables, and operators, which computes to a value.
- The computation is called an **evaluation**. For example, 5 * 10 evaluates to 50.
- Expressions can also contain variable.

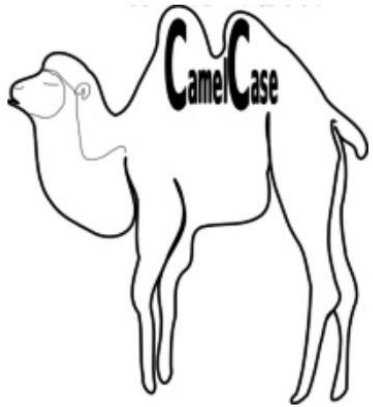
```
x * 10
```

JavaScript Identifiers

- Identifiers are names.
- In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).
- The rules for legal names are much the same in most programming languages.
- In JavaScript, the first character must be a **letter**, or an underscore (_), or a dollar sign (**$**).
- Subsequent characters may be letters, digits, underscores, or dollar signs.
- All JavaScript identifiers are **case sensitive**.

JavaScript and Camel Case

- Historically, programmers have used different ways of joining multiple words into one variable name:
  - **Hyphens**
    first-name, last-name, master-card, inter-city
  - **Underscore**
    first_name, last_name, master_card, inter_city
  - **Upper Camel Case (Pascal Case)**
    FirstName, LastName, MasterCard, InterCity
  - **Lower Camel Case**
    firstName, lastName, masterCard, interCity



- JavaScript programmers tend to use camel case that starts with a lowercase.

JS 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.

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).

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

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

Multi-line Comments

- Multi-line comments start with /* and end with */
- Any text between /* and */ will be ignored by JavaScript.

```
/*
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.";
```

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.

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

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

JS Variables

Value Data Types

Values assigned to variables fall under a few **data types**:

- **Undefined :**The variable is declared by giving it a name, but no value:

```
var foo;

alert(foo); // Will open a dialog containing "undefined"
```

- **`Null`** **:**Assigns the variable no inherent value:

```
var foo = null;

alert(foo); // Will open a dialog containing "null"
```

- **Numbers :**When you assign a number (e.g., 5), JavaScript treats it as a number (you don't need to tell it it's a number):

```
var foo = 5;

alert(foo + foo); // This will alert "10"
```

Value Data Types (cont'd)

- **Strings :** If the value is wrapped in single or double quotes, it is treated as a string of text:

```
var foo = "five";
alert(foo); // Will alert "five"
alert(foo + foo); // Will alert "fivefive"
```

- **Booleans :** Assigns a true or false value, used for scripting logic:

```
var foo = true; // The variable "foo" is now true
```

- **Arrays :** A group of multiple values (called *members*) assigned to a single variable. Values in arrays are *indexed* (assigned a number starting with 0). You can refer to array values by their index numbers:

```
var foo = [5, "five", "5"];

alert( foo[0] ); // Alerts "5"
alert( foo[1] ); // Alerts "five"
alert( foo[2] ); // Also alerts "5"
```

JavaScript Variables

- JavaScript variables are containers for storing data values.

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

- From the example above, you can expect:
  - x stores the value 5
  - y stores the value 6
  - z stores the value 11

Much Like Algebra

- 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).
- **JavaScript variables are containers for storing data values.**

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

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 _**
  - Names are **case sensitive** (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names
  - It may not contain special characters (! . , / \ + * =).

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. (The value of x is incremented by 5)

JavaScript Data Types

- JavaScript variables can hold numbers like 100 and text values like “Gaza".
- 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.

Declaring (Creating) JS 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";
```

Declaring (Creating) JS Variables

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

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

- Example:

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

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

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;
```

- A declaration can span multiple lines

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

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;
```

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;
```

JavaScript Arithmetic

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

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

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

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

JavaScript Arithmetic

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

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

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

JS Assignment

Name	Shorthand operator	Meaning
Assignment	`x = y`	`x = y`
Addition assignment	`x += y`	`x = x + y`
Subtraction assignment	`x -= y`	`x = x - y`
Multiplication assignment	`x *= y`	`x = x * y`
Division assignment	`x /= y`	`x = x / y`
Remainder assignment	`x %= y`	`x = x % y`
Exponentiation assignment	`x **= y`	`x = x ** y`
Left shift assignment	`x <<= y`	`x = x << y`
Right shift assignment	`x >>= y`	`x = x >> y`
Unsigned right shift assignment	`x >>>= y`	`x = x >>> y`
Bitwise AND assignment	`x &= y`	`x = x & y`
Bitwise XOR assignment	`x ^= y`	`x = x ^ y`
Bitwise OR assignment	`x |= y`	`x = x | y`

JS Data Types

JS 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
```

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";
```

- JavaScript will treat the example above as:

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

JS 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
```

JS Strings

- A string (or a text string) is a series of characters like "IUG".
- Strings are written with quotes.
- You can use single or double 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
```

JS Strings - Length

- The length of a string is found in the built-in property **length**.

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

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = txt.length;
```

JavaScript String Properties

The length property returns the length of a string:

26

JS Strings - Special Characters

- The **backslash** (\) escape character turns special characters into string characters.

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

JS Strings - Breaking Long Code Lines

- For best readability, programmers often like to avoid code lines longer than 80 characters.
- You can break up a code line within a text string with a **single backslash**.

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

- A safer way to break up a string, is to use string addition

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

JS Numbers

- Numbers can be written with, or without decimals.
- Extra large or extra small numbers can be written with scientific (exponential) notation.

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

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

JS Numbers - Types

- Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.
- JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.
- This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63.

Value (aka Fraction/Mantissa)	Exponent	Sign
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

JS Numbers - Numbers and Strings

- JavaScript uses the + operator for both addition and concatenation.
- Numbers are added. Strings are concatenated.

```
var x = 10;
var y = 20;
var z = x + y;            // z will be 30 (a number)

var x = "10";
var y = "20";
var z = x + y;           // z will be 1020 (a string)

var x = 10;
var y = "20";
var z = x + y;           // z will be 1020 (a string)
```

JS Numbers - Numbers and Strings

```
var x = "10";
var y = 20;
var z = x + y;            // z will be 1020 (a string)

var x = 10;
var y = 20;
var z = "The result is: " + x + y;

var x = 10;
var y = 20;
var z = "30";
var result = x + y + z;
```

JS Numbers - Numeric Strings

- JavaScript strings can have numeric content.
- JavaScript will try to convert strings to numbers in all numeric operations.

```
var x = "100";
var y = "10";
var z = x / y;       // z will be 10

var x = "100";
var y = "10";
var z = x * y;       // z will be 1000

var x = "100";
var y = "10";
var z = x - v;       // z will be 90
```

JS Numbers - NaN (Not a Number)

- **NaN** is a JavaScript reserved word indicating that a number is not a legal number.
- Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number).

```
var x = 100 / "Apple";  // x will be NaN (Not a Number)
```

- You can use the global JavaScript function **isNaN( )** to find out if a value is a number.

```
var x = 100 / "Apple";
isNaN(x);               // returns true because x is Not a Number
```

JS Numbers - Infinity

- Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

```
var myNumber = 2;
while (myNumber != Infinity) {          // Execute until Infinity
  myNumber = myNumber * myNumber;
}
```

- Division by 0 (zero) also generates Infinity.

```
var x =  2 / 0;          // x will be Infinity
var y = -2 / 0;          // y will be -Infinity
```

JS Booleans

- Booleans can only have two values: true or false.
- Booleans are often used in conditional testing.

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

JS Booleans - Boolean() Function

- You can use the Boolean() function to find out if an expression (or a variable) is true.

```
Boolean(10 > 9)          // returns true
```

- Or even easier:

```
(10 > 9)                 // also returns true
10 > 9                   // also returns true
```

JS Booleans - Comparisons

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

JS Booleans - Value

- Everything With a "Value" is True.
- Everything Without a "Value" is False.

```
100

3.14

-15

"Hello"

"false"
```

```
var x = 0;
Boolean(x);       // returns false
```

JS Arrays

- JavaScript arrays are written with square brackets.
- Array items are separated by commas.

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

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

JS Arrays

- You can change an array element using its index.

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

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

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

JavaScript Arrays

Saab,Volvo,BMW

JS Array 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". person[0]

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

- Objects use names to access its "members". person.firstName

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

JS Array Length

- The **length** property of an array returns the length of an array (the number of array elements).
- The length property is always one more than the highest array index.

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

Associative Arrays

- Many programming languages support arrays with **named indexes**.
- Associative arrays (dictionaries) that can hold data in (key, value) pairs.
- Arrays with named indexes are called **associative arrays**.
- JavaScript does **NOT** support arrays with named indexes.
- In JavaScript, arrays always use numbered indexes.
- 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.
- 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.

Associative Arrays

```
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
```

new Array() Constructor

- You can use the JavaScript's built-in array constructor **new Array()** to define a new array.
- 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
```

new Array() Constructor Problem

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


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

JS Arrays - toString()

- The JavaScript method **toString()** converts an array to a string of (comma separated) array values.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result

```
Banana,Orange,Apple,Mango
```

JS Arrays - join()

- The **join()** method also joins all array elements into a string.
- It behaves just like toString(), but in addition you can specify the separator.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result

```
Banana * Orange * Apple * Mango
```

JS Arrays - pop()

- The **pop()** method removes the last element from an array.
- The pop() method returns the value that was "popped out"

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();                 // Removes the last element ("Mango") from fruits
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();         // the value of x is "Mango"
```

JS Arrays - push()

- The **push()** method adds a new element to an array (at the end).
- The push() method returns the new array length.

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

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");    //  the value of x is 5
```

JS Arrays - shift()

- Shifting is equivalent to popping, working on the first element instead of the last.
- The **shift()** method removes the first array element and "shifts" all other elements to a lower index.
- The shift() method returns the string that was "shifted out"

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.shift();    // the value of x is "Banana"
```

JS Arrays - unshift()

- The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements.
- The unshift() method returns the new array length.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Returns 5
```

JS Arrays - Deleting Elements

- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete.
- Using delete may leave undefined holes in the array. Use pop() or shift() instead.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];           // Changes the first element in fruits to undefined
```

JS Arrays - splice()

- The **splice()** method can be used to add new items to an array.
- Parameters:
  - the **position** where new elements should be added
  - The **length** or how many elements should be removed
  - the new **elements** to be added
- The splice() method returns an array with the deleted items.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");

var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
```

JS Arrays - splice()

- With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);          // Removes the first element of fruits
```

JS Arrays - concat()

- The **concat()** method creates a new array by merging (concatenating) existing arrays.

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys
```

- The concat() method can take any number of array arguments

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);
```

JS Arrays - slice()

- The **slice()** method slices out a piece of an array into a new array.

Slice(from, to)

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

- The slice() method creates a new array. It does not remove any elements from the source array.

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3);
```

JS Arrays - sort()

- The **sort()** method sorts an array alphabetically.
- the sort() function sorts values as strings.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.sort();
  document.getElementById("demo").innerHTML = fruits;
}
```

Apple,Banana,Mango,Orange

JS Arrays - reverse()

- The **reverse()** method reverses the elements in an array.
- You can use it to sort an array in descending order.

```
// Create and display an array:
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  // First sort the array
  fruits.sort();
  // Then reverse it:
  fruits.reverse();
  document.getElementById("demo").innerHTML = fruits;
}
```

Orange,Mango,Banana,Apple

JS Objects

- Objects are variables. But they can contain many values.
- The values are written as **name:value** pairs (name and value separated by a colon :) and they called **properties**.
- You define (and create) a JavaScript object with an object literal

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

- Spaces and line breaks are not important. An object definition can span multiple lines.
- You can access object properties in two ways

    objectName.propertyName

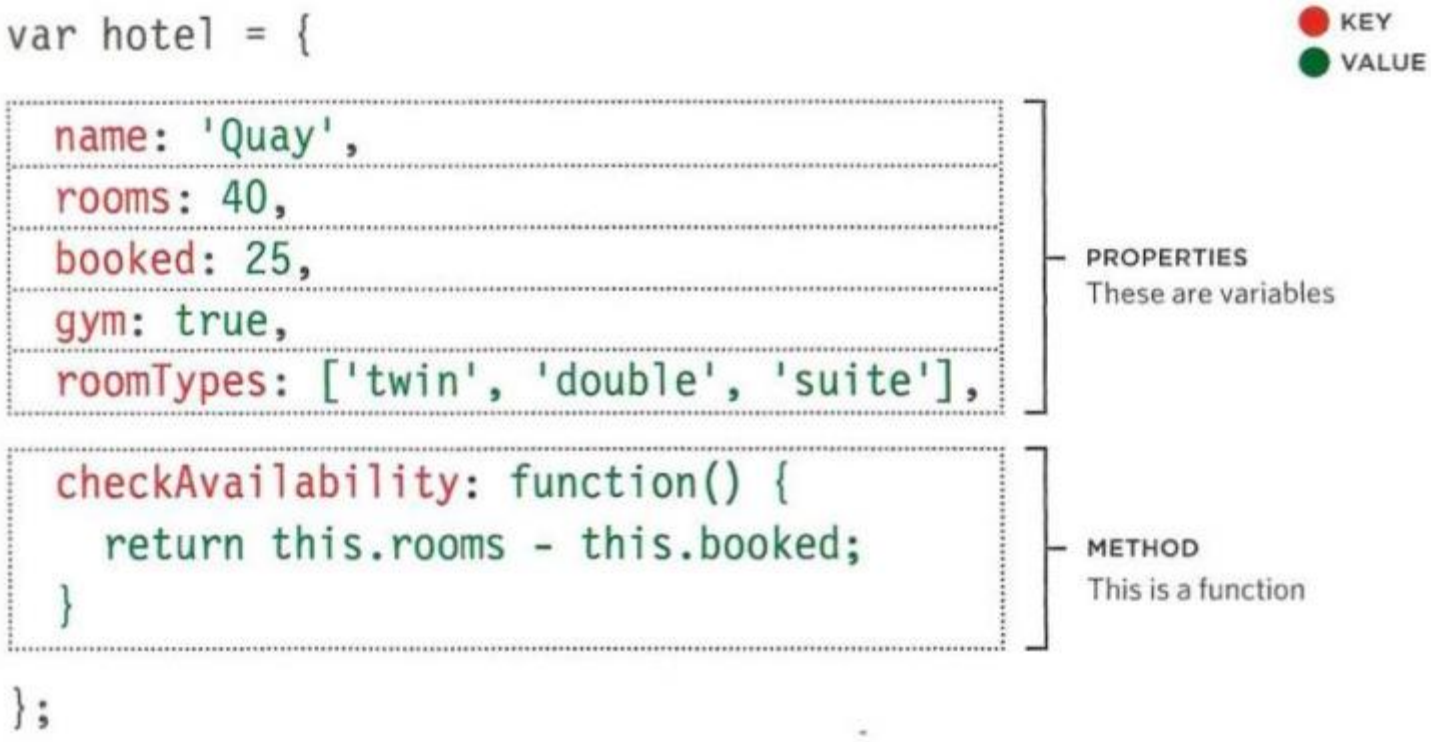
    objectName["propertyName"]

JS Objects

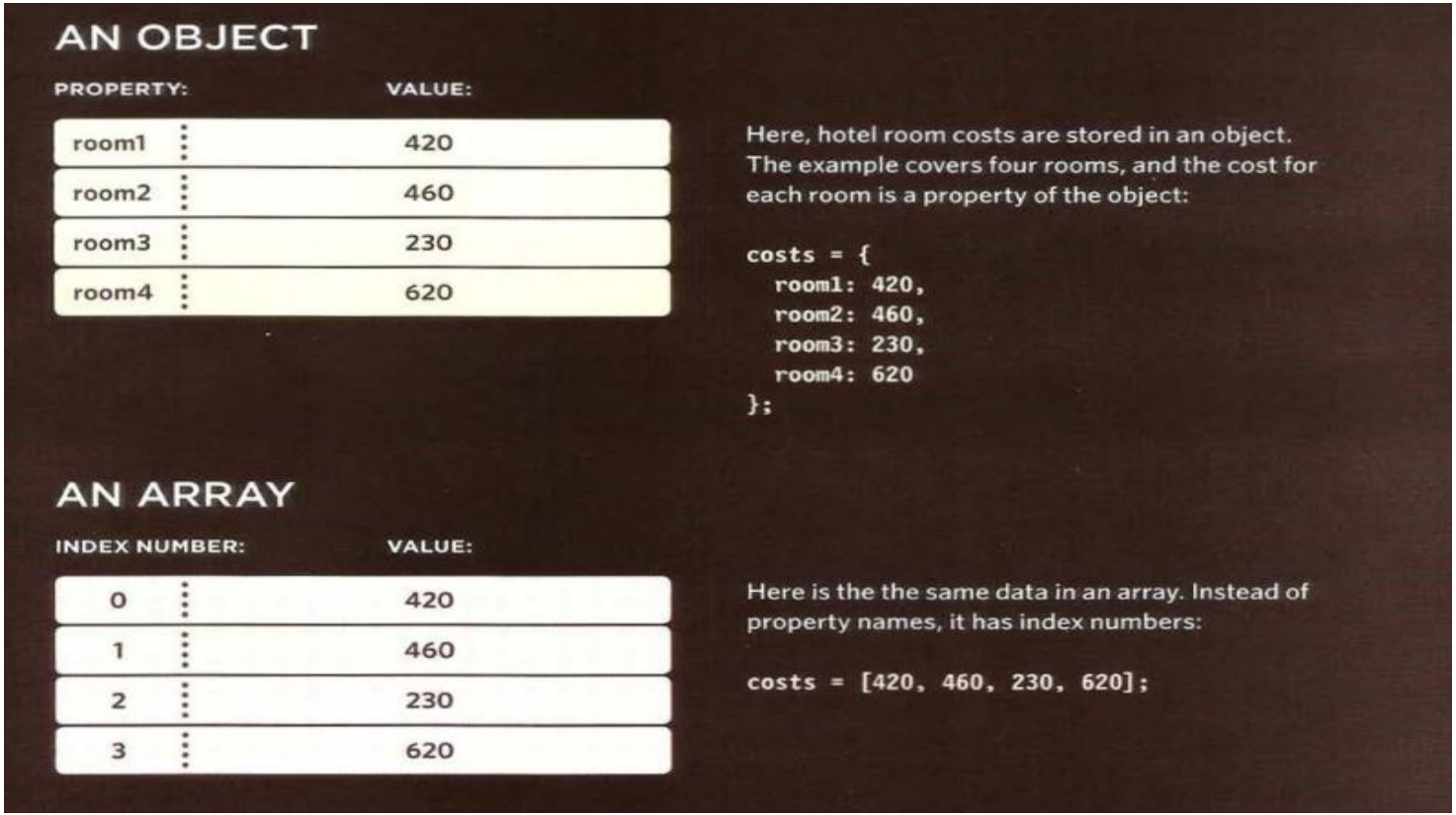
- Objects can also have **methods**. Methods are actions that can be performed on objects. Methods are stored in properties as function definitions.
- In a function definition, “**this**” refers to the "owner" of the function.

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

- You access an object method with the following syntax: objectName.methodName()


KEY
VALUE
var hotel = {
name: 'Quay',
rooms: 40,
booked: 25,
gym: true,
roomTypes: ['twin', 'double', 'suite'],
PROPERTIES
These are variables
checkAvailability: function() {
return this.rooms - this.booked;
}
METHOD
This is a function
};

Array vs object



JS Functions

JavaScript Function Definitions

- JavaScript functions are defined with the **function** keyword.
- You can use a **function declaration** or a **function expression**.
- Function Declarations:

```
function functionName(parameters) {
  // code to be executed
}
```

- Function Expressions (anonymous):

```
var x = function (a, b) {return a * b};
```

- Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

JavaScript Function Parameters

- JavaScript function definitions do not specify data types for parameters.
- JavaScript functions do not perform type checking on the passed arguments.
- JS functions do not check the number of arguments received. If a function is called with missing arguments (less than declared), the missing values are set to: **undefined**.

```
function myFunction(x, y) {
  if (y === undefined) {
    y = 0;
  }
}
```

The Arguments Object

- JavaScript functions have a built-in object called the **arguments** object.
- The argument object contains an array of the arguments used when the function was called (invoked).

```
x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {
  var i;
  var sum = 0;
  for (i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
```

Variable Scope

- A variable that can only be used within one function is **locally scoped**. When you define a variable inside a function, include the **`var`** keyword to keep it locally scoped (recommended):

```
var foo = "value";
```

- A variable that can be used by any script on your page is said to be **globally scoped**.
  - Any variable created outside of a function is automatically globally scoped:

```
var foo = "value";
```

    - To make a variable created inside a function globally scoped, omit the var keyword

```
foo = "value";
```

JS Math Object

Math Object

- The JavaScript **Math** object allows you to perform mathematical tasks on numbers.
- Unlike other global objects, the Math object **has no constructor**. Methods and properties are **static**.
- All methods and properties (constants) can be used without creating a Math object first.

Math Object Properties

- JavaScript provides 8 mathematical **constants** that can be accessed with the Math object.

```
Math.E          // returns Euler's number
Math.PI         // returns PI
Math.SQRT2      // returns the square root of 2
Math.SQRT1_2    // returns the square root of ½
Math.LN2        // returns the natural logarithm of 2
Math.LN10       // returns the natural logarithm of 10
Math.LOG2E      // returns base 2 logarithm of E
Math.LOG10E     // returns base 10 logarithm of E
```

Math Object Methods

- Math object contains many mathematical methods:

  abs(x) //Returns the absolute value of x

  ceil(x) //Returns the value of x rounded up to its nearest integer

  floor(x) //Returns the value of x rounded down to its nearest integer

  round(x) //Returns the value of x rounded to its nearest integer

  cos(x) //Returns the cosine of x (x is in **radians**)

  sin(x) //Returns the sine of x (x is in **radians**)

```
90 * Math.PI / 180
```

Math Object Methods

```
max(x, y, z, ..., n)      //Returns the number with the highest value

min(x, y, z, ..., n)      //Returns the number with the lowest value

random()                  //Returns a random number between 0 and 1

pow(x, y)                 //Returns the value of x to the power of y

log(x)                    //Returns the natural logarithm (base E) of x

sqrt(x)                   //Returns the square root of x
```

JS Conditions

Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- You can use conditional statements in your code to do this.
- In JavaScript we have the following conditional statements:
  - **if** to specify a block of code to be executed, if a specified condition is true
  - **else** to specify a block of code to be executed, if the same condition is false
  - **else if** to specify a new condition to test, if the first condition is false
  - **switch** to specify many alternative blocks of code to be executed

The if Statement

- Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

```
var hour = 11;
if (hour < 18) {
  document.getElementById("demo").innerHTML = "Good day!";
}
```

Display "Good day!" if the hour is less than 18:00:

Good day!

The else Statement

- Use the **else** statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

The else if Statement

- Use the **else if** statement to specify a new condition if the first condition is false.

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and
condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and
condition2 is false
}
```

The Switch Statement

- Use the **switch** statement to select one of many code blocks to be executed.
- Switch cases use strict comparison (===)

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

The Switch Statement - Examples

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

The Switch Statement - Examples

```
switch (today) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

JS Loops

JS Loops

- Loops can execute a block of code a number of times.
- JavaScript supports different kinds of loops:
  - **for** - loops through a block of code a number of times
  - **for/in** - loops through the properties of an object
  - **For/of**- loops through the properties of an array
  - **while** - loops through a block of code while a specified condition is true
  - **do/while** - also loops through a block of code while a specified condition is true

The For Loop

- The **for** loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed.

```
for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

The While Loop

- The **while** loop loops through a block of code as long as a specified condition is true.

```
while (condition) {
  // code block to be executed
}
```

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

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

The Do/While Loop

- The **do/while** loop is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {
  // code block to be executed
}
while (condition);
```

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

The Break Statement

- You have already seen the **break** statement used in this chapter. It was used to "jump out" of a **switch()** statement.
- The **break** statement can also be used to jump out of a loop.
- The **break** statement breaks the loop and continues executing the code after the loop.

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

The Continue Statement

- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

JS Events

JS Events

- HTML events are "things" that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can "react" on these events.
- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

```
<element event="some JavaScript">
```

JS Events

- Here is a list of some common HTML events:

onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

DIFFERENT EVENT TYPES

load	Web page has finished loading
unload	Web page is unloading (usually because a new page was requested)
error	Browser encounters a JavaScript error, or an asset doesn't exist
resize	Browser window has been resized
scroll	User has scrolled up or down the page

keydown	User first presses a key (repeats while key is depressed)
keyup	User releases a key
keypress	Character is being inserted (repeats while key is depressed)

DIFFERENT EVENT TYPES

click	User presses and releases a button over the same element
dblclick	User presses and releases a button twice over the same element
mousedown	User presses a mouse button while over an element
mouseup	User releases a mouse button while over an element
mousemove	User moves the mouse (not on a touchscreen)
mouseover	User moves the mouse over an element (not on a touchscreen)
mouseout	User moves the mouse off an element (not on a touchscreen)

DIFFERENT EVENT TYPES

FOCUS EVENTS — Occur when an element (e.g., a link or form field) gains or loses focus

EVENT	DESCRIPTION
`focus / focusin`	Element gains focus
`blur / focusout`	Element loses focus

FORM EVENTS — Occur when a user interacts with a form element

EVENT	DESCRIPTION
`input`	Value in any `<input>` or `<textarea>` element has changed (IE9+) or any element with the `contenteditable` attribute
`change`	Value in select box, checkbox, or radio button changes (IE9+)
`submit`	User submits a form (using a button or a key)
`reset`	User clicks on a form's reset button (rarely used these days)
`cut`	User cuts content from a form field
`copy`	User copies content from a form field
`paste`	User pastes content into a form field
`select`	User selects some text in a form field

HTML EVENT HANDLER ATTRIBUTES (DO NOT USE)

This method of event handling is no longer used because it is better to separate the JavaScript from the HTML

```
<form method="post" action="http://www.example.org/register">
  <label for="username">Create a username: </label>
  <input type="text" id="username" onblur="checkUsername()" />
  <div id="feedback"></div>

  <label for="password">Create a password: </label>
  <input type="password" id="password" />

  <input type="submit" value="Sign up!" />
</form>
...
<script type="text/javascript" src="js/event-attributes.js"></script>
```

```
function checkUsername() {                                    // Declare function
  var elMsg = document.getElementById('feedback');            // Get feedback element
  var elUsername = document.getElementById('username');// Get username input
  if (elUsername.value.length < 5) {                          // If username too short
    elMsg.textContent = 'Username must be 5 characters or more'; // Set msg
  } else {                                                     // Otherwise
    elMsg.textContent = '';                                    // Clear message
  }
}
```

TRADITIONAL DOM EVENT HANDLERS

- you can only attach one function to each event handler

```
element.onevent = functionName;
```

ELEMENT	EVENT	CODE
DOM element node to target	Event bound to node(s) preceded by word "on"	Name of function to call (with no parentheses following it)

A reference to the DOM element node is often stored in a variable.

```
function checkUsername() {
  // code to check the length of username
}
var el = document.getElementById('username');
el.onblur = checkUsername;
```

The event name is preceded by the word "on."

The code starts by defining the named function.

The function is called by the event handler on the last line, but the parentheses are omitted.

TRADITIONAL DOM EVENT HANDLERS

JAVASCRIPT

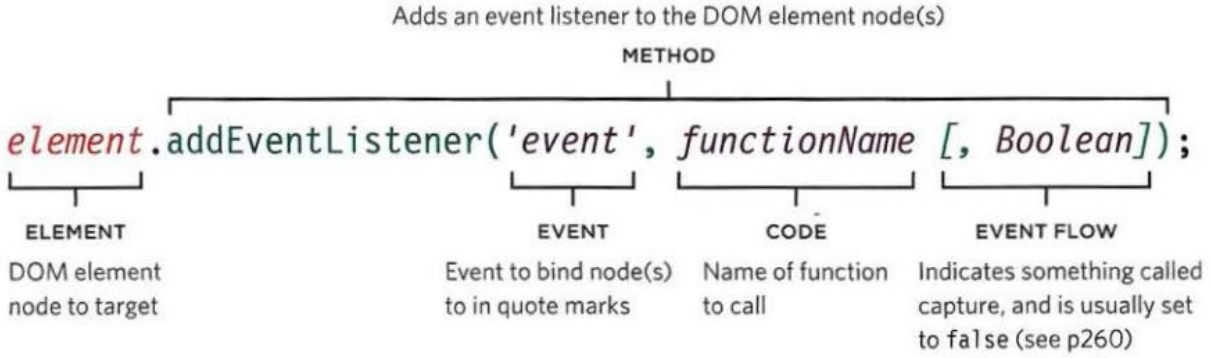
c06/js/event-handler.js

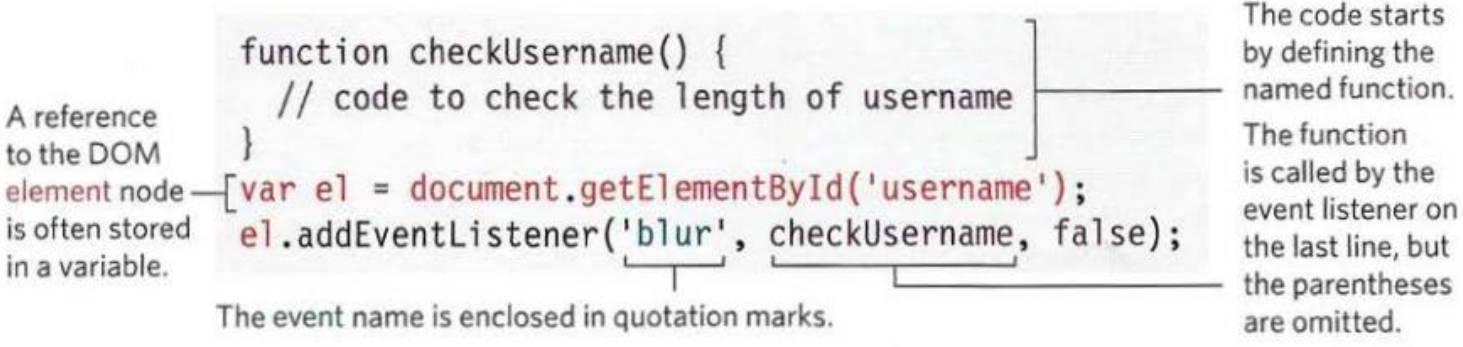
```
function checkUsername() {                                  // Declare function
  var elMsg = document.getElementById('feedback');          // Get feedback element
  if (this.value.length < 5) {                              // If username too short
    elMsg.textContent = 'Username must be 5 characters or more';  // Set msg
  } else {                                                  // Otherwise
    elMsg.textContent = '';                                 // Clear message
  }
}

var elUsername = document.getElementById('username');  // Get username input
elUsername.onblur = checkUsername;  // When it loses focus call checkuserName()
```

① ② ③

EVENT LISTENERS

- They can deal with more than one function at a time





EVENT LISTENERS

Function is written first

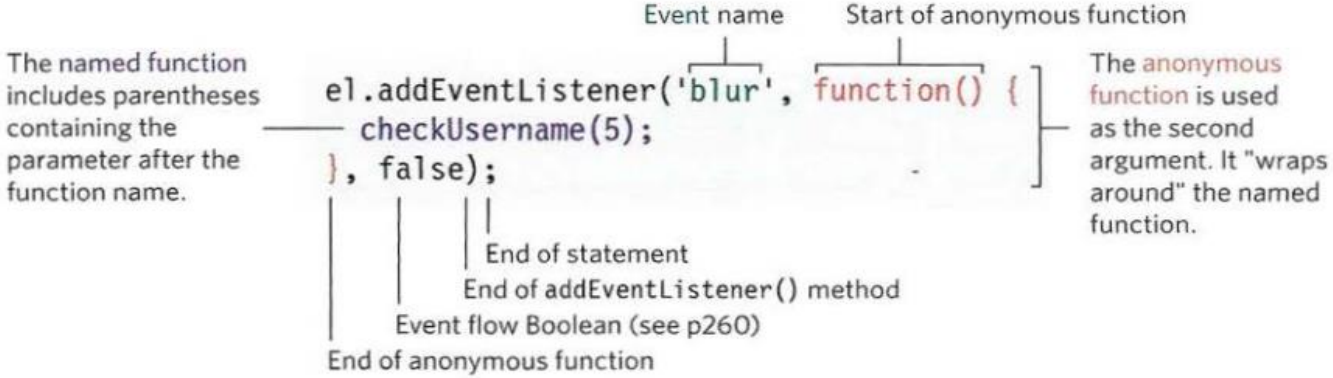
JAVASCRIPT

c06/js/event-listener.js

```
function checkUsername() {                                    // Declare function
  var elMsg = document.getElementById('feedback');            // Get feedback element
  if (this.value.length < 5) {                                // If username too short
    elMsg.textContent = 'Username must be 5 characters or more'; // Set msg
  } else {                                                    // Otherwise
    elMsg.textContent = '';                                   // Clear msg
  }
}

var elUsername = document.getElementById('username');  // Get username input
// When it loses focus call checkUsername()
elUsername.addEventListener('blur', checkUsername, false);
```

① ②

USING PARAMETERS WITH EVENT LISTENERS




```
var elUsername = document.getElementById('username');  // Get username input
var elMsg = document.getElementById('feedback');       // Get feedback element

function checkUsername(minLength) {                    // Declare function
  if (elUsername.value.length < minLength) {           // If username too short
   // Set the error message
   elMsg.textContent = 'Username must be ' + minLength + ' characters or more';
  } else {                                             // Otherwise
   elMsg.innerHTML = '';                               // Clear msg
  }
}

elUsername.addEventListener('blur', function() {       // When it loses focus
  checkUsername(5);                                     // Pass arguments here
}, false);
```

onchange Event

- The **onchange** event occurs when the value of an element has been changed.
- For radiobuttons and checkboxes, the onchange event occurs when the checked state has been changed.

```
Enter your name: <input type="text" id="fname"
onchange="myFunction()">

<script>
function myFunction() {
  var x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>
```

onclick Event

- The **onclick** event occurs when the user clicks on an element.

```
<p id="demo" onclick="myFunction()">Click me.</p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "YOU CLICKED
ME!";
}
```

onclick Event

```
<p id="demo" onclick="myFunction()">Click me to change my text
color.</p>

<script>
function myFunction() {
  document.getElementById("demo").style.color = "red";
}
</script>
```

onmouseover / onmouseout Event

- The **onmouseover** event occurs when the mouse pointer is moved onto an element, or onto one of its children.
- This event is often used together with the **onmouseout** event, which occurs when a user moves the mouse pointer out of an element.

onmouseover / onmouseout Event

```
<h1 id="demo" onmouseover="mouseOver()"
onmouseout="mouseOut()">Mouse over me</h1>

<script>
function mouseOver() {
  document.getElementById("demo").style.color = "red";
}

function mouseOut() {
  document.getElementById("demo").style.color = "black";
}
</script>
```

onkeydown Event

- The **onkeydown** event occurs when the user is pressing a key (on the keyboard).

```
<input type="text" id="demo" onkeydown="myFunction()">

<script>
function myFunction() {
  document.getElementById("demo").style.backgroundColor =
"red";
}
</script>
```

onkeyup Event

- The **onkeyup** event occurs when the user releases a key (on the keyboard).

```
<input type="text" id="demo" onkeydown="keydownFunction()"
onkeyup="keyupFunction()">

<script>
function keydownFunction() {
  document.getElementById("demo").style.backgroundColor =
"red";
}

function keyupFunction() {
  document.getElementById("demo").style.backgroundColor =
"green";
}
</script>
```

onload Event

- The **onload** event occurs when an object has been loaded.
- onload is most often used within the **<body>** element to execute a script once a web page has completely loaded all content (including images, script files, CSS files, etc.).
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- The onload event can also be used to deal with cookies (see "More Examples" below).

onload Event

```
<!DOCTYPE html>
<html>
<body onload="whenPageLoaded()">

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

<script>
function whenPageLoaded() {
  alert("Page is Loaded !");
}
</script>

</body>
</html>
```

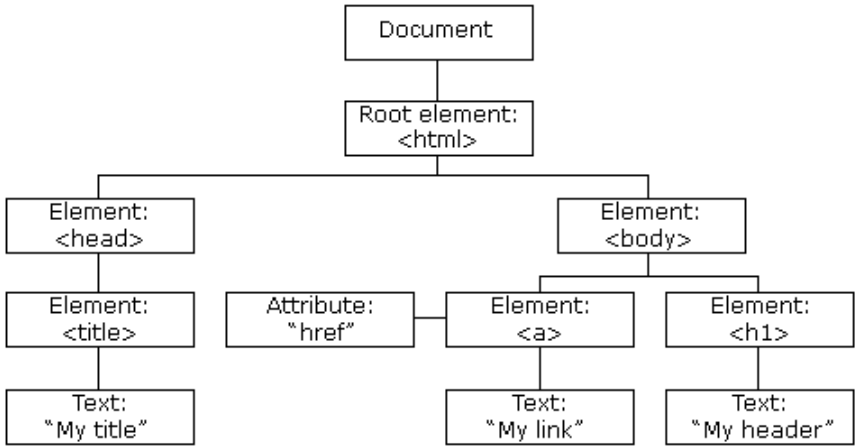
onload Event

```
<iframe onload="myFunction()" src="index.php"></iframe>

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

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Iframe is
loaded.";
}
</script>
```

JS HTML DOM

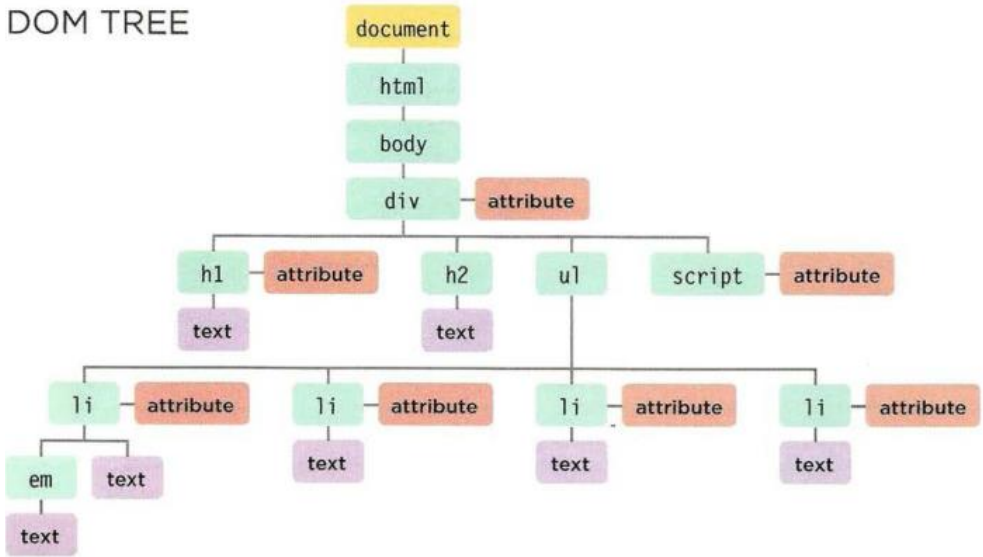
Introduction

- With the HTML DOM (Document Object Model), JavaScript can access and change all the elements of an HTML document.
- When a web page is loaded, the browser creates a Document Object Model of the page.
- The HTML DOM model is constructed as a tree of Objects:



Tree code

```
<html>
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
      <script src="js/list.js"></script>
    </div>
  </body>
</html>
```



What is HTML DOM?

- The HTML DOM is a standard object model and programming interface for HTML. It defines:
  - The HTML **elements** as objects
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements
- In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

DOM Methods

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

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

- In the example above, **getElementById** is a **method**, while **innerHTML** is a **property**.

The HTML DOM Document Object

- The **document** object represents your web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.
- On next slide, you can see some examples of how you can use the document object to access and manipulate HTML.

The HTML DOM Document Object

- **Finding HTML Elements:**
- **document.getElementById(id)** Find an element by element id
- **document.getElementsByTagName(<li>)** Find element**s** [] by tag name
- **document.getElementsByClassName(classname)** Find element**s** [] by class name

- **Changing HTML Elements:**
- **element.innerHTML** Change the inner HTML of an element
- **element.attribute** Change the attribute value of an element
- **element.setAttribute(att, val)** Change the attribute value of an element
- **element.style.property** Change the style of an HTML element

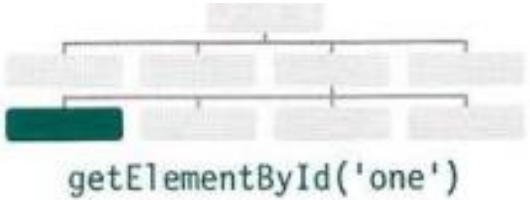
DOM Elements

- With JavaScript, you want to manipulate HTML elements.
- You have saw how to find the elements. There are many ways to do this:
  - Finding HTML elements by id
  - Finding HTML elements by tag name
  - Finding HTML elements by class name
  - Finding HTML elements by CSS selectors
  - Finding HTML elements by HTML object collections

ACCESS THE ELEMENTS

SELECT AN INDIVIDUAL **ELEMENT** NODE

- getElementByld (‘id’) Selects an individual element given the value of its id attribute.

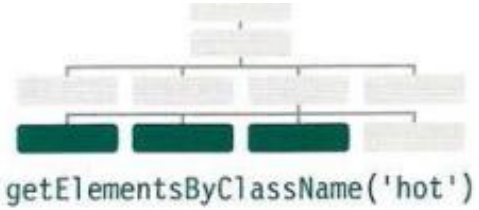


- querySelector(‘ css selector ') Uses a CSS selector and returns the first matching element.

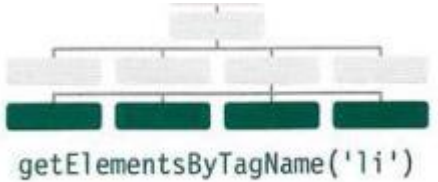
ACCESS THE ELEMENTS

.SELECT MULTIPLE ELEMENTS (NODELISTS)

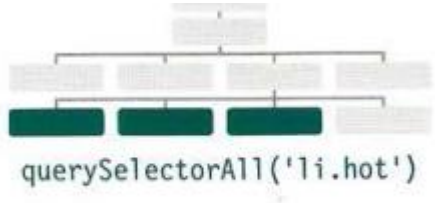
- getElementsByClassName(‘class ‘) Selects all elements that have a specific value for their class attribute.



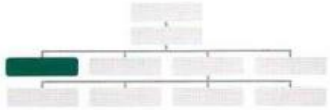
- getElementsByTagName(‘ tagName ‘) Selects all elements that have the specified tag name. (faster than querySelectorAll())



- querySelectorAll (‘css selector ‘) Uses CSS selector syntax to select one or more elements and returns all of those that match



getElementsByTagName('h1')



Even though this query only returns one element, the method still returns a NodeList because of the potential for returning more than one element.

INDEX NUMBER & ELEMENT
0 <h1>

getElementsByTagName('li')

This method returns four elements, one for each of the <li> elements on the page. They appear in the same order as they do in the HTML page.

INDEX NUMBER & ELEMENT
0 <li id="one" class="hot">
1 <li id="two" class="hot">
2 <li id="three" class="hot">
3 <li id="four">

getElementsByClassName('hot')

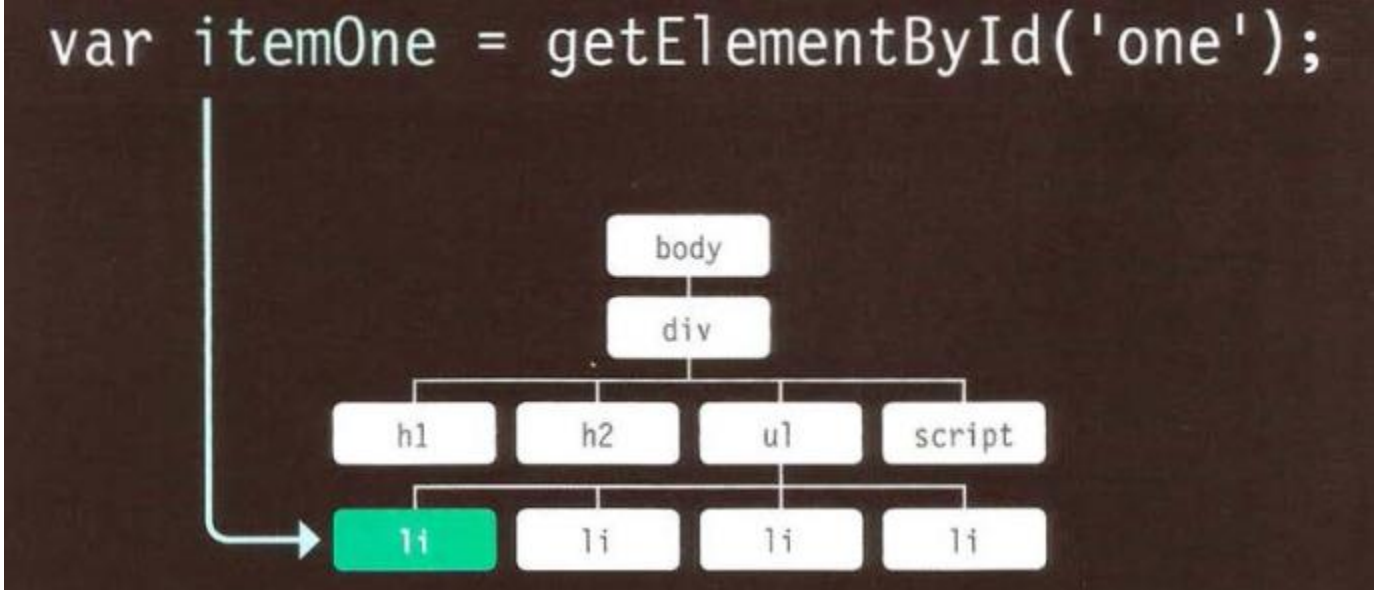
This NodeList contains only three of the <li> elements because we are searching for elements by the value of their class attribute, not tag name.

INDEX NUMBER & ELEMENT
0 <li id="one" class="hot">
1 <li id="two" class="hot">
2 <li id="three" class="hot">

querySelectorAll('li[id]')

This method returns four elements, one for each of the <li> elements on the page that have an id attribute (regardless of the values of the id attributes).

INDEX NUMBER & ELEMENT
0 <li id="one" class="hot">
1 <li id="two" class="hot">
2 <li id="three" class="hot">
3 <li id="four">



HTML

c05/get-element-by-id.html

```
<h1 id="header">List King</h1>
<h2>Buy groceries</h2>
<ul>
  <li id="one" class="hot"><em>fresh</em>
    figs</li>
  <li id="two" class="hot">pine nuts</li>
  <li id="three" class="hot">honey</li>
  <li id="four">balsamic vinegar</li>
</ul>
```

JAVASCRIPT

c05/js/get-element-by-id.js

```
// Select the element and store it in a variable.
var el = document.getElementById('one');

// Change the value of the class attribute.
el.className = 'cool';
```

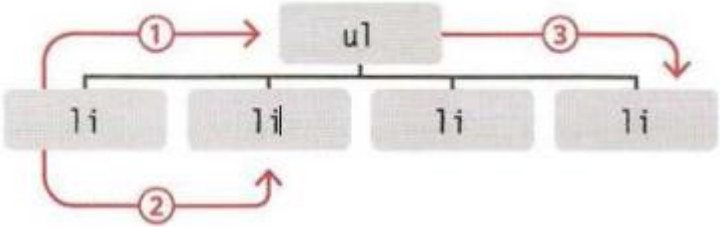
```
var hotItems = document.querySelectorAll('li.hot');
for (var i = 0; i < hotItems.length; i++) {
  hotItems[i].className = 'cool';
}
```

JAVASCRIPT

c05/js/node-list.js

```
var hotItems = document.querySelectorAll('li.hot'); // Store NodeList in array

if (hotItems.length > 0) {                          // If it contains items

    for (var i=0; i<hotItems.length; i++) {     // Loop through each item
        hotItems[i].className = 'cool';     // Change value of class attribute
    }
```

ACCESS THE ELEMENTS



TRAVERSING BETWEEN ELEMENT NODES

1. parentNode Selects the parent of the current element node (which will return just one element)
2. previousSibling / nextSibling Selects the previous or next sibling from the DOM tree
3. firstChild / lastChild Select the content first or last child of the current element.

- All these are properties not functions
- If no next sibling/first/last child result is null

Previous/nextSibling

- nextSibling/previousSibling return next, previous siblings if not white space between them
- previousElementSibling /nextElementSibling return previous, next sibling even if they have space between them

c05/sibling.html

```
<ul><li id="one" class="hot"><em>fresh</em> figs</li><li id="two"
class="hot">pine nuts</li><li id="three" class="hot">honey</li><li
id="four">balsamic vinegar</li></ul>
```

c05/js/sibling.js

```
// Select the starting point and find its siblings
var startItem = document.getElementById('two');
var prevItem = startItem.previousSibling;
var nextItem = startItem.nextSibling;

// Change the values of the siblings' class attributes
prevItem.className = 'complete';
nextItem.className = 'cool';
```

First/last child

- `First/lastchild` give different result if space were between parent and child
- `firstElementChild`/ `lastElementChild` can be used for more accuracy

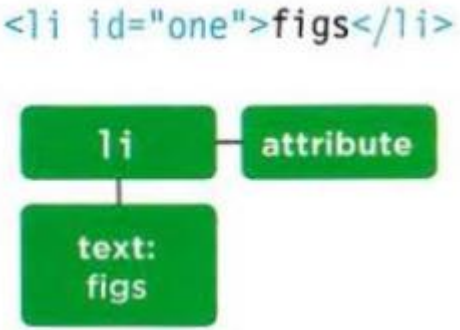
HTML

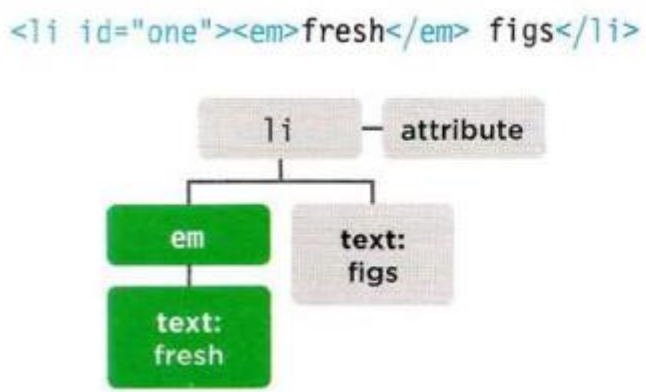
```
<ul
  ><li id="one" class="hot"><em>fresh</em> figs</li
  ><li id="two" class="hot">pine nuts</li
  ><li id="three" class="hot">honey</li
  ><li id="four">balsamic vinegar</li
></ul>
```

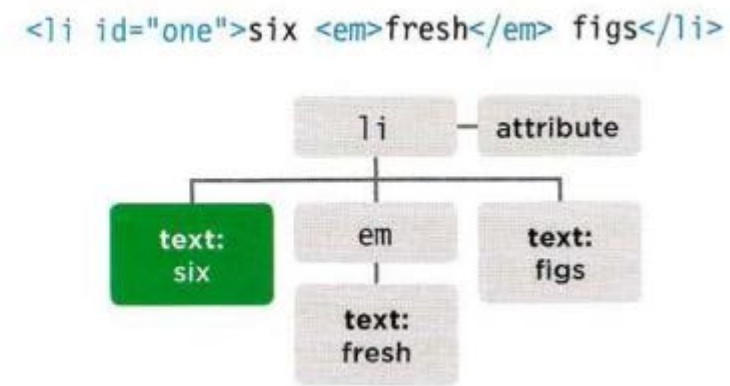
JAVASCRIPT

```
// Select the starting point and find its children
var startItem = document.getElementsByTagName('ul')[0];
var firstItem = startItem.firstChild;
var lastItem = startItem.lastChild;

// Change the values of the children's class attributes
firstItem.setAttribute('class', 'complete');
lastItem.setAttribute('class', 'cool');
```

GET/UPDATE ELEMENT CONTENT

- 

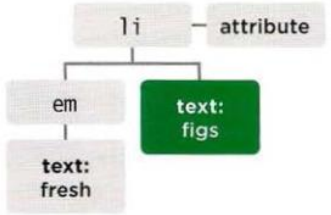




GET/UPDATE ELEMENT CONTENT

- **nodeValue** Accesses text from node
- **You must be on a text node, not the element that contains the text**

```
<li id="one"><em>fresh</em> figs</li>
```



The code below shows how you access the second text node. It will return the result: figs

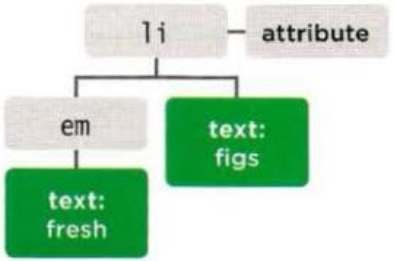
```
document.getElementById('one').firstChild.nextSibling.nodeValue;
```

① ② ③ ④

```
var itemTwo = document.getElementById('two');       // Get second list item

var elText  = itemTwo.firstChild.nodeValue;         // Get its text content

elText = elText.replace('pine nuts', 'kale');       // Change pine nuts to kale

itemTwo.firstChild.nodeValue = elText;              // Update the list item
```

GET/UPDATE ELEMENT CONTENT

```
<li id="one"><em>fresh</em> figs</li>
```



textContent Gets/sets text only

- collect the text from the elements and ignore any markup inside the element.
- In this case it would return the value: fresh figs.

```
document.getElementById('one').textContent;
```

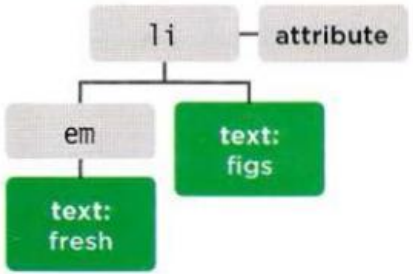
innerText Gets/sets text only

- Same as textContent but three differences
  - Not supported by many browsers
  - It will not show any content that has been hidden by CSS.
  - it can be slower to retrieve the content than the textContent property.

GET/UPDATE ELEMENT CONTENT

- **innerHTML**: access and amend the contents of an element, including any child elements
- Get elements content as string
- To update element content, it takes string that might contain markup
-

```
<li id="one"><em>fresh</em> figs</li>
```



GET CONTENT

The following line of code collects the content of the list item and adds it to a variable called `elContent`:

```
var elContent = document.getElementById('one').innerHTML;
```

The `elContent` variable would now hold the string:

```
'<em>fresh</em> figs'
```

SET CONTENT

The following line of code adds the content of the `elContent` variable (including any markup) to the first list item:

```
document.getElementById('one').innerHTML = elContent;
```

GET/UPDATE ELEMENT CONTENT

innerHTML

JAVASCRIPT

c05/js/inner-html.js

```
// Store the first list item in a variable
var firstItem = document.getElementById('one');

// Get the content of the first list item
var itemContent = firstItem.innerHTML;

// Update the content of the first list item so it is a link
firstItem.innerHTML = '<a href=\"http://example.org\">' + itemContent + '</a>';
```

Note how the quotes are escaped.

If you use attributes in your HTML code, escaping “ using \ can make it clearer that those characters are no part of the script

ADDING ELEMENTS

1. createElement () : create element
2. createTextNode() : create text
3. appendChild(): join parent with child

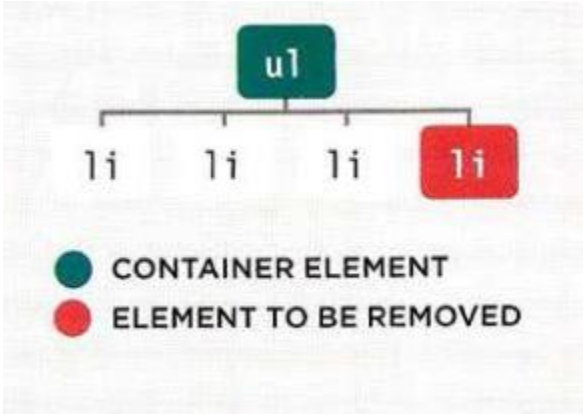
JAVASCRIPT

```
// Create a new element and store it in a variable.
var newEl = document.createElement('li');

// Create a text node and store it in a variable.
var newText = document.createTextNode('quinoa');

// Attach the new text node to the new element.
newEl.appendChild(newText);

// Find the position where the new element should be added.
var position = document.getElementsByTagName('ul')[0];

// Insert the new element into its position.
position.appendChild(newEl);
```

RESULT

REMOVING ELEMENTS

1. Access element need to be removed
2. Access parent of element
3. Parent.removeChild(element)




```
var removeEl = document.getElementsByTagName('li')[3]; // The element to remove

var containerEl = removeEl.parentNode;                  // Its containing element

containerEl.removeChild(removeEl);                      // Removing the element
```

ATTRIBURES NODES

METHOD	DESCRIPTION
`getAttribute()`	gets the value of an attribute
`hasAttribute()`	checks if element node has a specified attribute
`setAttribute()`	sets the value of an attribute
`removeAttribute()`	removes an attribute from an element node

PROPERTY	DESCRIPTION
`className`	gets or sets the value of the **`class`** attribute
`id`	gets or sets the value of the **`id`** attribute

CHECK ATTRIBUTE & GET ITS VALUES

```
var firstItem = document.getElementById('one');     // Get first list item

if (firstItem.hasAttribute('class')) {              // If it has class attribute
    var attr = firstItem.getAttribute('class');     // Get the attribute

    // Add the value of the attribute after the list
    var el = document.getElementById('scriptResults');
    el.innerHTML = '<p>The first item has a class name: ' + attr + '</p>';

}
```

CREATE/CHANHE ATTRIBUTES

```
var firstItem = document.getElementById('one'); // Get the first item
firstItem.className = 'complete';               // Change its class attribute

var fourthItem = document.getElementsByTagName('li').item(3);// Get fourth item
el2.setAttribute('class', 'cool');              // Add an attribute to it
```

- className: allows to change class of element or add it if not exist
- The setAttribute() method allows you to update the value of *any* attribute.

REMOVING ATTRIBUTES

```
var firstItem = document.getElementById('one'); // Get the first item
if (firstItem.hasAttribute('class')) {          // If it has a class attribute
    firstItem.removeAttribute('class');         // Remove its class attribute
}
```

- Removing attribute that not exist does not cause an error

Changing HTML Style

- The HTML DOM allows JavaScript to change the style of HTML elements.
- To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = new style
```

```
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>
```

End

Front End