

JavaScript: Client-Side Scripting

Abdallah Karakra & Sobhi Ahmed

Chapter 6

Objectives

1 What is **JavaScript**

2 JavaScript **Design**

3 Using
JavaScript

4 Syntax

5 JavaScript
Objects

6 The DOM

7 JavaScript
Events

8 **Forms**

Section 1 of 8

WHAT IS JAVASCRIPT

What is JavaScript

- JavaScript runs right **inside the browser**
- JavaScript is **dynamically typed**
- JavaScript is object oriented in that almost **everything in the language is an object**
 - the objects in JavaScript are prototype-based rather than class-based, which means that while JavaScript shares some syntactic features of PHP, Java or C#, it is also quite **different from those languages**

What isn't JavaScript

It's not Java

Although it contains the word *Java*, JavaScript and Java are vastly **different programming languages** with different uses. Java is a full-fledged compiled, object-oriented language, popular for its ability to **run on any platform with a JVM installed**.

Conversely, JavaScript is one of the world's most popular languages, with fewer of the object-oriented features of Java, and **runs directly inside the browser, without the need for the JVM**.

Client-Side Scripting

It's good

There are many **advantages** of client-side scripting:

- Processing can be offloaded from the server to client machines, thereby **reducing the load on the server**.
- The browser can **respond more rapidly to user events** than a request to a remote server ever could, which improves the **user experience**.

Client-Side Scripting

There are challenges

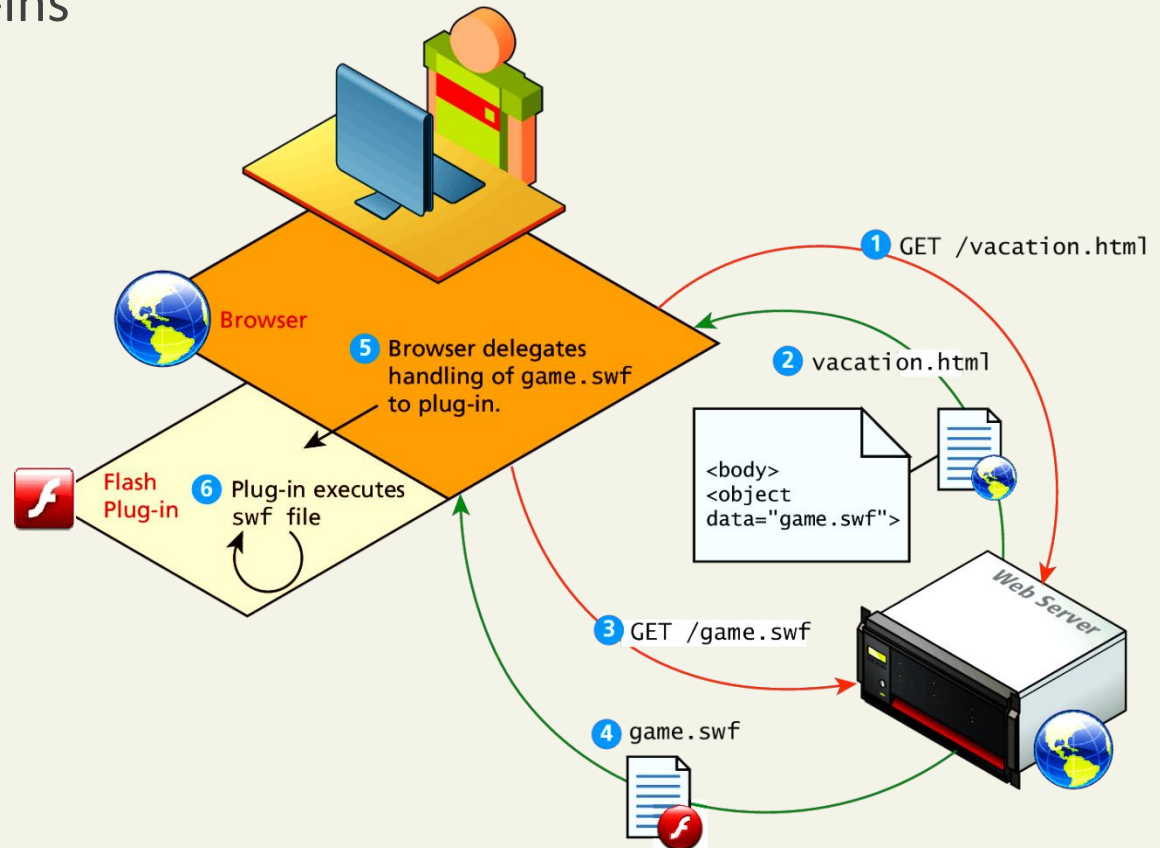
The **disadvantages** of client-side scripting are mostly related to how programmers use JavaScript in their applications.

- There is **no guarantee that the client has JavaScript enabled**
- **What works in one browser, may generate an error in another.**
- **Complicated to debug and maintain.**

Client-Side Flash

JavaScript is not the only type of client-side scripting.

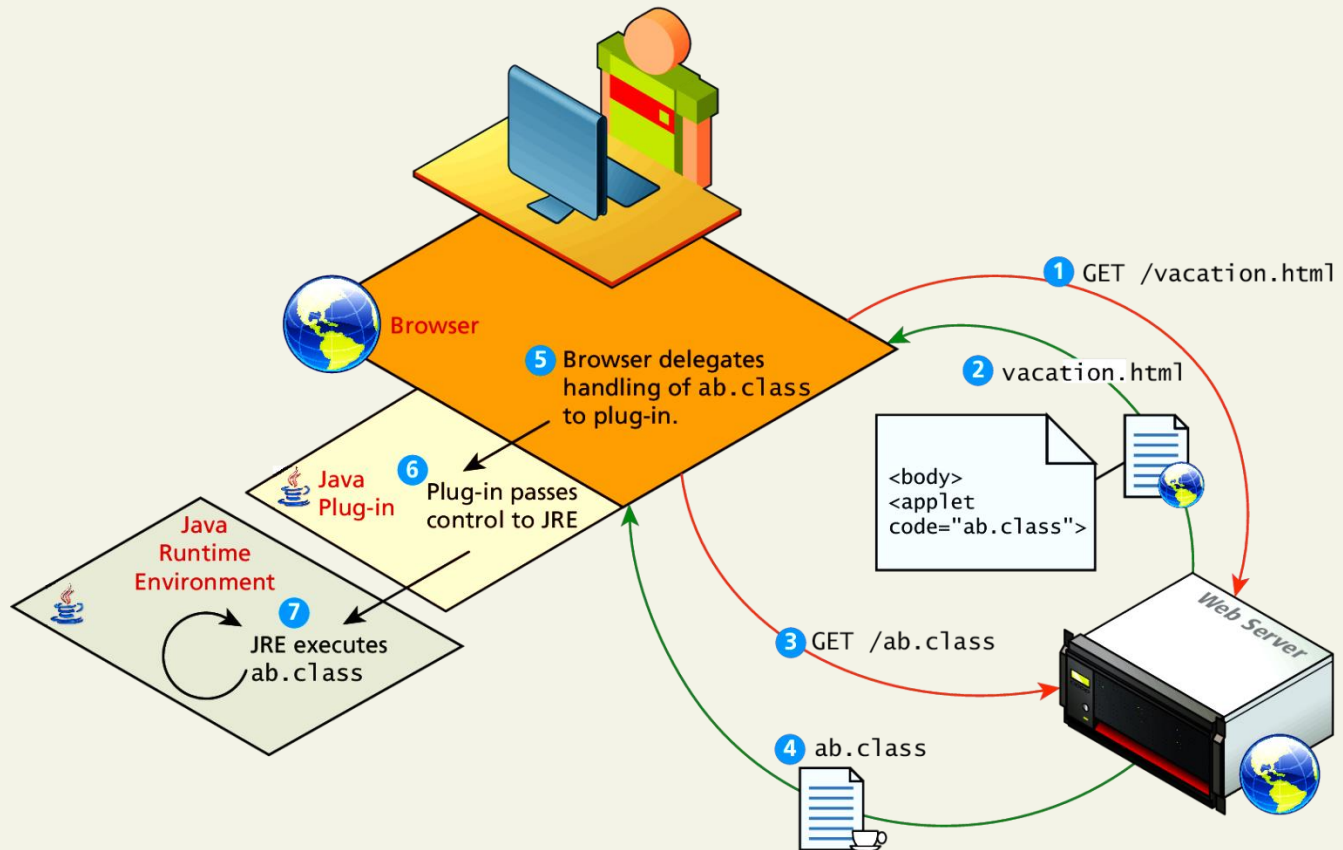
- Browser Plug-ins
 - Flash



Client-Side Applets

Java Applets

Java applets are written in and are separate objects included within an HTML document via the `<applet>` tag



JavaScript History

- JavaScript was introduced by **Netscape** in their **Navigator browser back in 1996**.
- JavaScript is in fact an implementation of a standardized scripting language called **ECMAScript**

JavaScript in Modern Times

AJAX

AJAX is both an acronym as well as a general term.

- As an acronym it means **A**synchronous JavaScript **A**nd **X**ML.

The `<noscript>` tag

Mechanism to speak to those with JavaScript

Any text between the opening and closing tags will only be **displayed to users without the ability to load JavaScript.**

It is often **used to prompt users to enable JavaScript,** but can also be used to **show additional text to search engines.**

Requiring JavaScript (or Flash) for the basic operation of your site will cause problems eventually and should be avoided.

The `<noscript>` tag defines an alternate content for users that have disabled scripts in their browser or have a browser that doesn't support script.

The `<noscript>` tag

Mechanism to speak to those with JavaScript

```
<body>  
  
<script>  
document.write("Hello World!")  
</script>  
<noscript>Sorry, your browser does not support JavaScript!</noscript>  
  
<p>A browser without support for JavaScript will show the text inside the  
noscript element.</p>  
  
</body>
```

Hello World!

A browser without support for JavaScript will show the text inside the noscript element.

Section 3 of 8

WHERE DOES JAVASCRIPT GO?

Where does JavaScript go?

JavaScript can be linked to an HTML page in a number of ways.

- Inline
- Embedded
- External

Inline JavaScript

Mash it in

Inline JavaScript refers to the practice of including **JavaScript code directly within certain HTML attributes**

Inline JavaScript is a real maintenance nightmare

Code

```
<a href="JavaScript:OpenWindow();"more info</a>  
<input type="button" onclick="alert('Are you sure?');" />
```

LISTING 6.1 Inline JavaScript example

Embedded JavaScript

Better

Embedded JavaScript refers to the practice of **placing JavaScript code within a <script> element**

Code

```
<script type="text/javascript">
/* A JavaScript Comment */
alert ("Hello World!");
</script>
```

LISTING 6.2 Embedded JavaScript example

External JavaScript

Better

JavaScript supports this **separation** by allowing links to **an external file that contains the JavaScript.**

By convention, JavaScript external files have the extension **.js**.

Code

```
<head>
  <script type="text/JavaScript" src="greeting.js">
  </script>
</head>
```

LISTING 6.3 External JavaScript example

Section 4 of 8

SYNTAX

JavaScript Syntax

We will briefly cover the fundamental syntax for the most common programming constructs including

- **variables,**
- **assignment,**
- **conditionals,**
- **loops, and**
- **arrays**

before moving on to advanced topics such as **events** and **classes**.

Variables

var

Variables in JavaScript are **dynamically typed**, meaning a **variable can be an integer, and then later a string, then later an object**, if so desired.

This simplifies variable declarations, so that we do not require the familiar type fields like *int*, *char*, and *String*. Instead we use **var**

Assignment can happen at declaration-time by appending the value to the declaration, or at **run time** with a simple right-to-left assignment

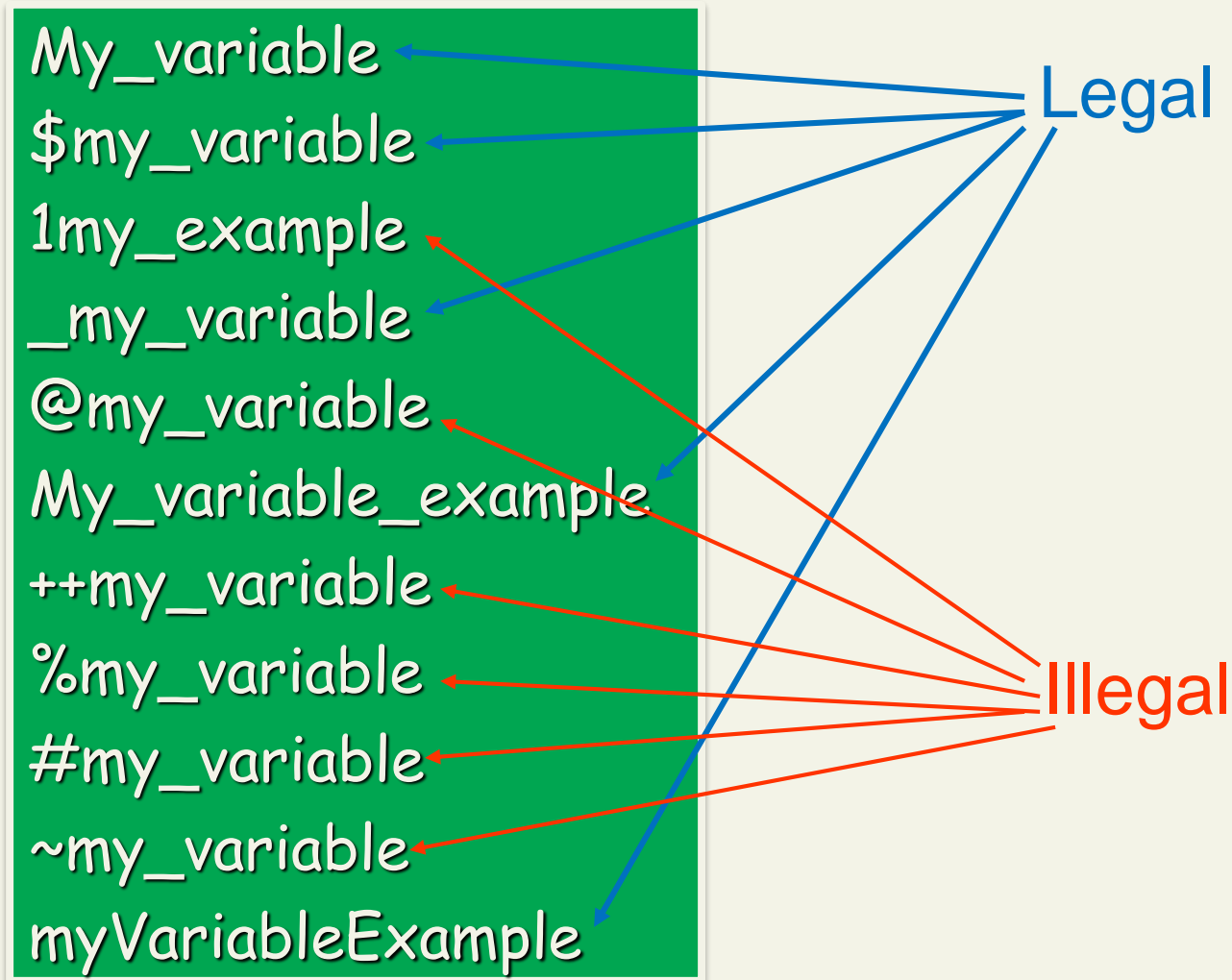
Variables

➤ JavaScript allows you to declare and use variables to store values.

➤ **How to assign a name to a variable?**

- Include uppercase and lowercase letters
- Digits from 0 through 9
- The underscore `_` and the dollar sign `$`
- No space and punctuation characters
- First character must be alphabetic letter or underscore
- Case-sensitive
- No reserved words or keywords

Which one is legal ?



Variables and Data Types

Example variable declarations and Assignments

Defines a variable named `abc`

```
var abc;
```

Each line of JavaScript should be terminated with a semicolon

```
var def = 0;
```

A variable named `def` is defined and initialized to `0`

```
def= 4 ;
```

`def` is assigned the value of `4`

Notice that whitespace is unimportant

```
def =  
"hello" ;
```

`def` is assigned the value of `"hello"`

Notice that a line of JavaScript can span multiple lines

Variables and Data Types

Data Types

two basic data types:

- **reference types** (usually referred to as objects) and
- **primitive types**

Primitive types represent simple forms of data.

- **Boolean, Number, String, ...**

Variables and Data Types

Reference Types

```
var abc = 27;  
var def = "hello";
```

variables with primitive types

```
var foo = [45, 35, 25];
```

variable with reference type
(i.e., array object)

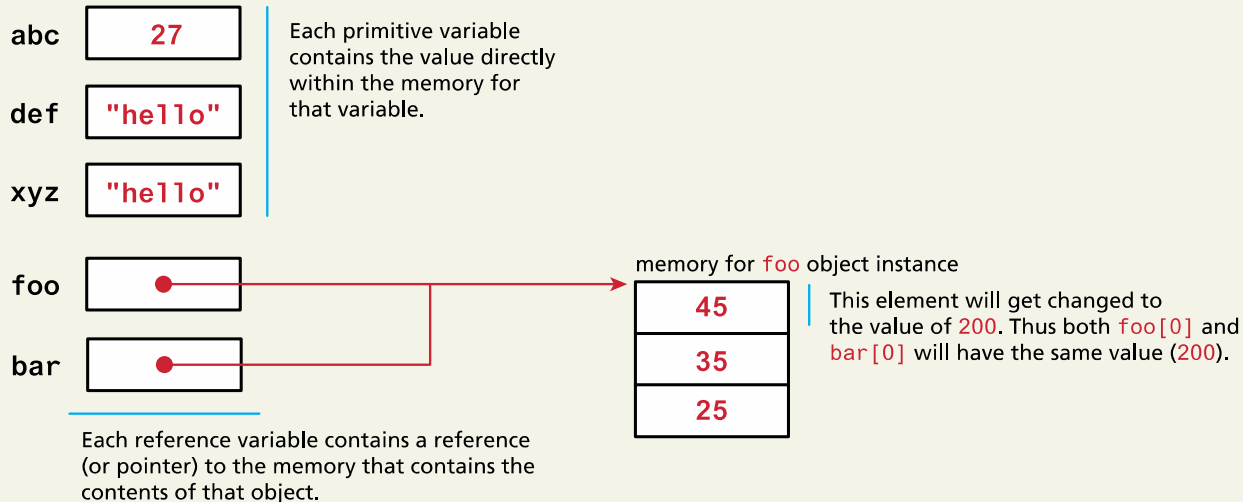
```
var xyz = def;  
var bar = foo;
```

these new variables differ in important ways
(see below)

```
bar[0] = 200;
```

changes value of the first element of array

Memory representation



JavaScript Output

```
var name = "Randy";  
  
document.write("<h1>Title</h1>");  
  
// this uses the concatenate operator (+)  
  
document.write("Hello " + name + " and welcome");
```

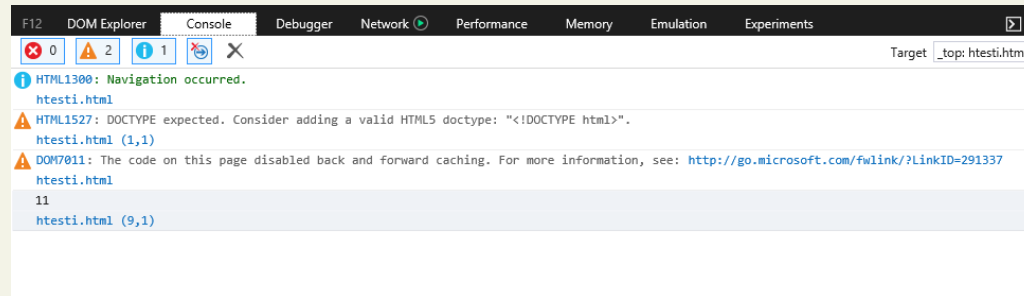
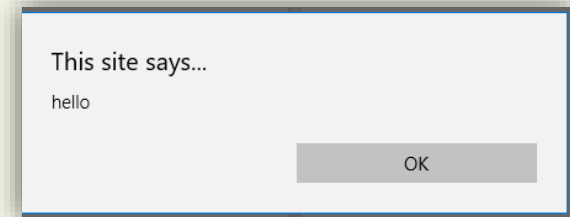
```
<!DOCTYPE html>  
<html>  
<head>  
<title>Page Title</title>  
</head>  
<body>  
<script type="text/javascript">  
  var name = "Randy";  
  document.write("<h1>Title</h1>");  
  // this uses the concatenate operator (+)  
  document.write("Hello " + name + " and welcome");  
</script>  
  
</body>  
</html>
```

Title

Hello Randy and welcome

JavaScript Output

- `alert()` Displays content within a **pop-up box**.
- `console.log()` Displays content in the Browser's **JavaScript console**.



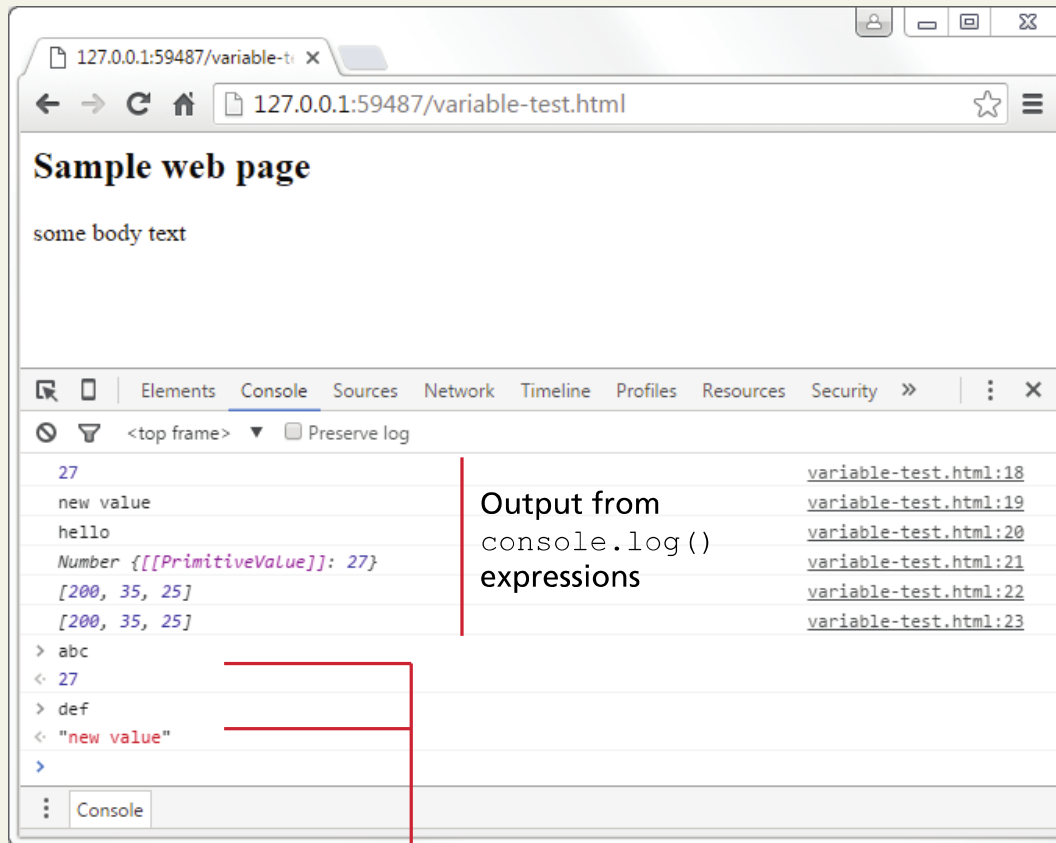
- `document.write()` Outputs the content (as markup) directly **to the HTML document**.



JavaScript Output

Chrome JavaScript Console

Web page content



JavaScript console

Output from
`console.log()`
expressions

Using console interactively to query
value of JavaScript variables

Chapter 8

1

JavaScript 1:
Language
Fundamentals

2

Where Does
JavaScript Go?

3

Variables and
Data Types

4

JavaScript
Output

5

Conditionals

6

Loops

7

Arrays

8

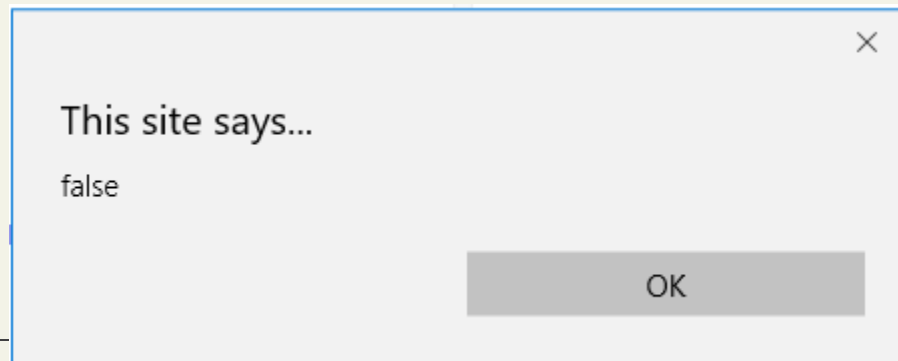
Objects

Comparison Operators

True or not True

Operator	Description	Matches (x=9)
==	Equals	(x==9) is true (x=="9") is true
===	Exactly equals, including type	(x==="9") is false (x===9) is true
<, >	Less than, Greater Than	(x<5) is false
<=, >=	Less than or equal, greater than or equal	(x<=9) is true
!=	Not equal	(4!=x) is true
!==	Not equal in either value or type	(x!== "9") is true (x!==9) is false

```
var x=2;  
var y="2";  
alert(x===y);
```



Logical Operators

The Boolean operators and, or, and not and their **truth tables** are listed in Table 6.2. Syntactically they are represented with && (and), || (or), and ! (not).

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND Truth Table

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

OR Truth Table

A	! A
T	F
F	T

NOT Truth Table

TABLE 6.2 AND, OR, and NOT Truth Tables

Conditionals

If, else if, ..., else

JavaScript's syntax is **almost identical to that of PHP, Java, or C** when it comes to conditional structures such as if and if else statements. In this syntax the condition to test is contained within () brackets with the body contained in { } blocks.

```
var hourOfDay; // var to hold hour of day, set it later...
var greeting; // var to hold the greeting message.
if (hourOfDay > 4 && hourOfDay < 12){
    // if statement with condition
    greeting = "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 20){
    // optional else if
    greeting = "Good Afternoon";
}
else{ // optional else branch
    greeting = "Good Evening";
}
```

Code

LISTING 6.4 Conditional statement setting a variable based on the hour of the day

Conditionals

switch

```
switch (type) {  
    case "PT":  
        output = "Painting";  
        break;  
    case "DO":  
        output = " Doctor ";  
        break;  
    default:  
        output = "Other";  
}
```

Conditionals

Conditional Assignment

```
/* x conditional assignment */  
x = (y==4) ? "y is 4" : "y is not 4";
```

<u>Condition</u>	<u>Value if true</u>	<u>Value if false</u>
------------------	--------------------------	---------------------------

```
/* equivalent to */  
if (y==4) {  
    x = "y is 4";  
}  
else {  
    x = "y is not 4";  
}
```

Conditionals

Truthy and Falsy

In JavaScript, a value is said to be **truthy** if it translates to true, while a value is said to be **falsy** if it translates to false.

- Almost all values in JavaScript are truthy
- **false, null, "", "", 0, NaN, and undefined are falsy**

Chapter 8

1

JavaScript 1:
Language
Fundamentals

2

Where Does
JavaScript Go?

3

Variables and
Data Types

4

JavaScript
Output

5

Conditionals

6

Loops

7

Arrays

8

Objects

Loops

While and do . . . while Loops

```
var count = 0;

while (count < 10) {

    // do something

    // ...

    count++;

}
```

```
count = 0;

do {

    // do something

    // ...

    count++;

} while (count < 10);
```

Loops

For Loops

initialization condition post-loop operation

```
for (var i = 0; i < 10; i++) {  
    // do something with i  
    // ...  
}
```

Chapter 8

1

JavaScript 1:
Language
Fundamentals

2

Where Does
JavaScript Go?

3

Variables and
Data Types

4

JavaScript
Output

5

Conditionals

6

Loops

7

Arrays

8

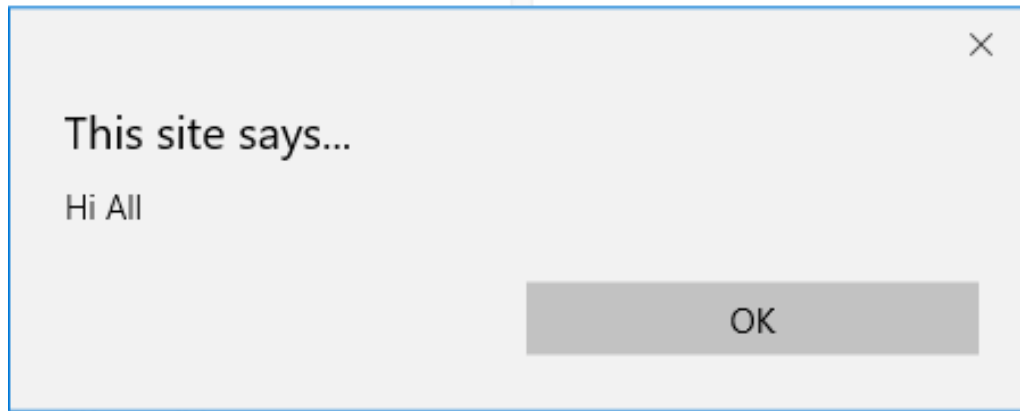
Objects

A JavaScript statement: alert

a JS command that pops up a dialog box with a message

```
alert("message");
```

```
alert("Hi All");
```



Arrays

Arrays are one of the most commonly used data structures in programming.

JavaScript provides two main ways to define an array.

- **object literal notation**
- use the **Array()** constructor

Arrays

The following code creates a new, **empty array** named greetings:

```
var greetings = new Array(); //length=0
```

Arrays

Initialize with values

To initialize the array with values, the variable declaration would look like the following:

```
var greetings = new Array("Good Morning", "Good Afternoon");//length=2
```

or, using the square bracket notation:

```
var greetings = ["Good Morning", "Good Afternoon");// array literal notation
```

Arrays

object literal notation

The literal notation approach is generally **preferred** since it involves **less typing**, is more **readable**, and executes a little **bit quicker**

```
var years = [1855, 1648, 1420];
```

```
var countries = ["Canada", "France",  
                "Germany", "Nigeria",  
                "Thailand", "United States"];
```

```
var mess = [53, "Canada", true, 1420];
```

Arrays

Access and Traverse

To access an element in the array you use the familiar square bracket notation from Java and C-style languages, with the index you wish to access inside the brackets.

```
alert ( greetings[0] );
```

One of the most common actions on an array is to traverse through the items sequentially. Using the Array object's **length** property to determine the maximum valid index. We have:

```
for (var i = 0; i < greetings.length; i++){  
    alert(greetings[i]);  
}
```

Arrays

Modifying an array

To **add an item to an existing array**, you can use the **push** method.

```
greetings.push("Good Evening");
```

The **pop ()** method can be used to **remove an item from the back of an array**.

Additional methods:

concat(), slice(), join(), reverse(), shift(), and sort()

Arrays

Modifying an array

Before

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var x= fruits.pop();// remove the last item of an array
```

After

```
fruits// Banana, Orange, Apple  
x// Mango
```

```
fruits.push("Mango");
```

```
fruits // Banana, Orange , Apple, Mango
```


Arrays

```
var a1 = ["rami", "khaled"];  
var a2 = ["sandy", "ali", "lina"];  
var a3 = a1.concat(a2);
```

```
//a3: rami Khaled sandy ali lina
```

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

//slice method selects the elements starting at the given *start* argument, and ends at, *but does not include*, the given *end* argument (method returns the selected elements in an array, as a new array object.).

```
//newArray: Orange,Lemon
```

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

```
fruits.shift(); // method removes the first item of an array.
```

```
// Orange , Apple, Mango
```

Functions

Functions are the building block for modular code in JavaScript, and are even used to build **pseudo-classes**, which you will learn about later.

They are defined by using the reserved word **function** and then the **function name and (optional) parameters**.

Since JavaScript is dynamically typed, **functions do not require a return type, nor do the parameters require type**.

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

JavaScript Functions

Defining functions

```
function name() {  
  statement;  
  statement;  
  ...  
  statement;  
}
```

```
function myFunction() {  
  alert("Hello!");  
  alert("How are you?");  
}
```

Functions

Example

Therefore a function to raise x to the yth power might be defined as:

```
function power(x,y){  
    var pow=1;  
    for (var i=0;i<y;i++){  
        pow = pow*x;  
    }  
    return pow;  
}
```

And called as

```
power(2,10);
```

Alert

Not really used anymore, console instead

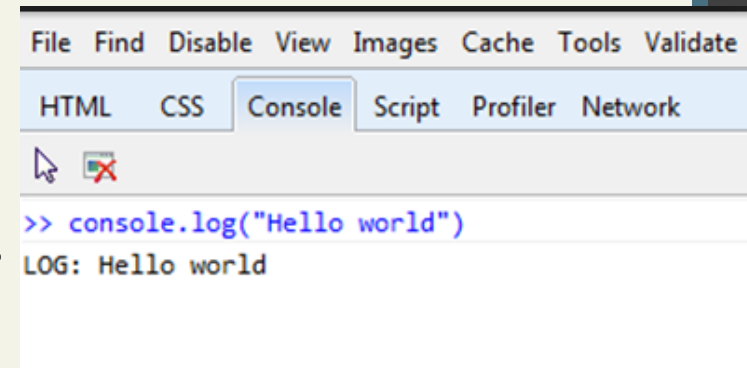
The `alert()` function makes the browser show a pop-up to the user, with whatever is passed being the message displayed. The following JavaScript code displays a simple hello world message in a pop-up:

```
alert ( "Good Morning" );
```

Using alerts can get tedious fast. When using debugger tools in your browser you can write output to a log with:

```
console.log("Put Messages Here");
```

And then use the debugger to access those logs.



Errors using try and catch

When the browser's JavaScript engine encounters an error, it will *throw* an **exception**. These exceptions interrupt the regular, sequential execution of the program and can stop the JavaScript engine altogether. However, you can optionally **catch these errors preventing disruption of the program** using the **try-catch block**

```
try {  
    nonexistentfunction("hello");  
}  
catch(err) {  
    alert("An exception was caught:" + err);  
}
```

LISTING 6.5 Try-catch statement

Throw your own

Exceptions that is.

Although try-catch can be used exclusively to catch built-in JavaScript errors, it can also be used by your programs, to throw your own messages. The throw keyword stops normal sequential execution, just like the built-in exceptions

```
try {
  var x = -1;
  if (x<0)
    throw "smallerthan0Error";
}
catch(err){
  alert (err + "was thrown");
}
```

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

LISTING 6.6 Throwing a user-defined exception

Tips

With Exceptions

Try-catch and throw statements should be used for *abnormal or exceptional cases in your program.*

Throwing an exception disrupts the sequential execution of a program. When the exception is thrown all subsequent code is not executed until the catch statement is reached.

This reinforces why try-catch is for exceptional cases.

Section 5 of 8

JAVASCRIPT OBJECTS

JavaScript Objects

Objects not Classes

JavaScript is not a full-fledged object-oriented programming language.

it does not support many of the patterns you'd expect from an object-oriented language like inheritance and polymorphism.

The language does, however, support objects.

JavaScript Objects

Not full-fledged O.O.

There are objects that are included in the JavaScript language; you can also define your own kind of objects.

Constructors

- create a new object we use the **new keyword**, the **class name**, and **() brackets with n optional parameters inside**, comma delimited as follows:

```
var someObject = new ObjectName(p1,p2,..., pn);
```

For some classes, **shortcut constructors** are defined

```
var greeting = "Good Morning";
```

vs the **formal**:

```
var greeting = new String("Good Morning");
```

Properties

Use the dot

Each object might have properties that can be accessed, depending on its definition.

When a property exists, it can be accessed using **dot notation** where a dot between the instance name and the property references that property.

```
//show someObject.property to the user  
alert(someObject.property);
```

Methods

Use the dot, with brackets

Objects can also have methods, which are **functions** associated with an instance of an object. These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method.

```
someObject.doSomething();
```

Methods may produce different output depending on the object they are associated with because *they can utilize the internal properties of the object.*

Objects Included in JavaScript

A number of useful objects are included with JavaScript including:

- Array
- Boolean
- Date
- Math
- String
- Dom objects

Math

[Reference List](#)

The **Math class** allows one to access common mathematic functions and common values quickly in one place.

This static class contains methods such as `max()`, `min()`, `pow()`, `sqrt()`, and `exp()`, and trigonometric functions such as `sin()`, `cos()`, and `arctan()`.

Many mathematical constants are defined such as `PI`, `E`, `SQRT2`, and some others

```
Math.PI; // 3.141592657
```

```
Math.sqrt(4); // square root of 4 is 2.
```

```
Math.random(); // random number between 0 and 1
```


Math

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
```

```
var three = Math.floor(Math.PI);
```

[Reference List](#)

String

The **String** class has already been used without us even knowing it.

Constructor usage

```
var greet = new String("Good"); // long form constructor
```

```
var greet = "Good"; // shortcut constructor
```

Length of a string

```
alert (greet.length); // will display "4"
```

String

Concatenation and so much more

```
var str = greet.concat("Morning"); // Long form concatenation
```

```
var str = greet + "Morning"; // + operator concatenation
```

Many other useful methods exist within the String class, such as

- accessing a single character using `charAt()`
- searching for one using `indexOf()`.

Strings allow splitting a string into an array, searching and matching with `split()`, `search()`, and `match()` methods.

Date

The **Date class** is yet another helpful included object you should be aware of. It allows you to quickly calculate **the current date** or **create date objects for particular dates**.

To display today's date as a string, we would simply create a new object and use the `toString()` method.

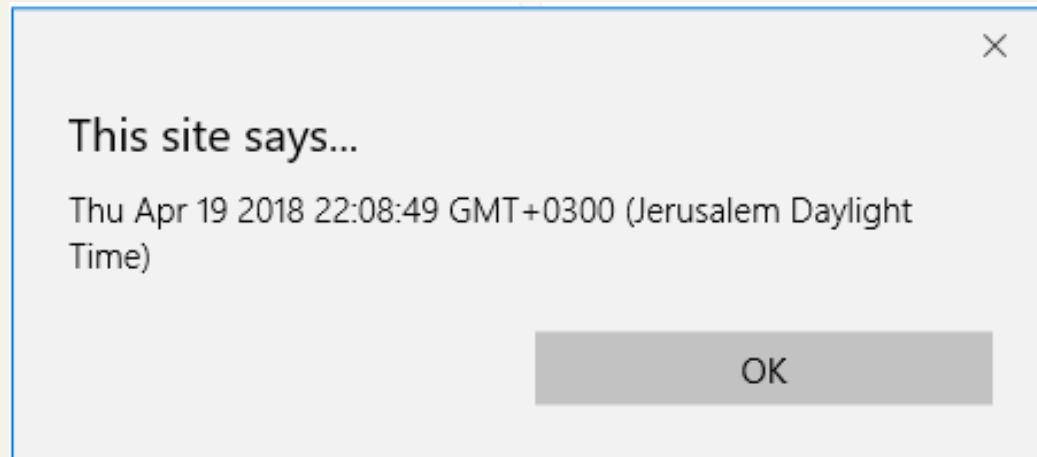
```
var d = new Date();
```

```
// This outputs Today is Mon Nov 12 2012 15:40:19 GMT-0700
```

```
alert ("Today is "+ d.toString();)
```

Date

```
<script>  
var obj= new Date();  
alert(obj.toString());  
</script>
```

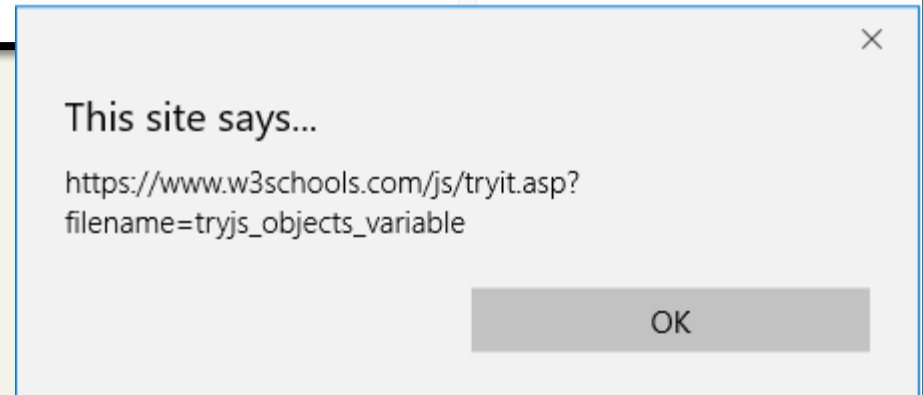


Window

The **window object** in JavaScript corresponds to the browser itself. Through it, you can access the **current page's URL**, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows.

In fact, the **alert()** function mentioned earlier is actually a **method of the window object**.

```
<script>
var currentLocation = window.location;
alert(currentLocation );
</script>
```



Window

The window object in JavaScript corresponds to the browser itself. Through it, you can access the current page's URL, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows.

In fact, the **alert()** function mentioned earlier is actually a method of the window object.

Other window functions:

window.alert()

window.prompt()

window.confirm()

window.prompt()

- To ask the user a question and **give the user a way to respond**, we could use the prompt method of the window object:
window.prompt(question,default);
- The method takes two arguments: the question to ask the user and a default answer, both string type values. We separate arguments with a comma.

window.prompt()

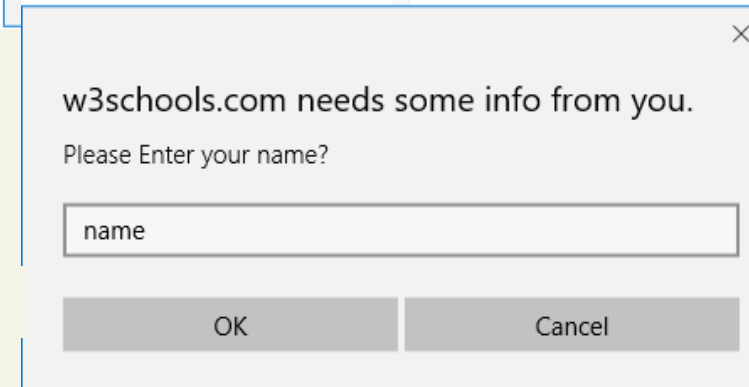
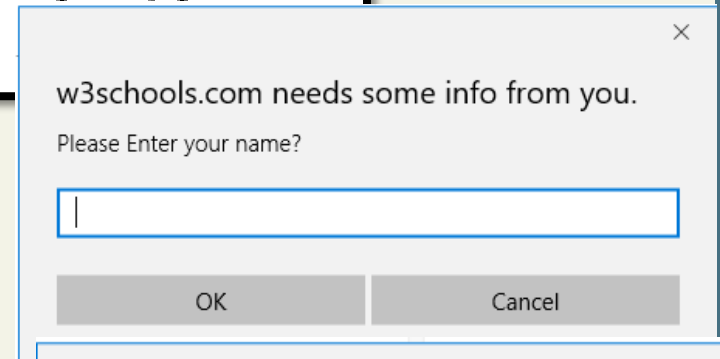
Prompt example

- We capture a user's answer to a question posed by window.prompt() by assigning the method to a variable:
`var answer = window.prompt(...);`
- The resulting value is always a string data-type.

```
<script>  
window.prompt("Please Enter your name?", "");  
</script>
```

```
<script>  
window.prompt("Please Enter your name?", "name");  
</script>
```

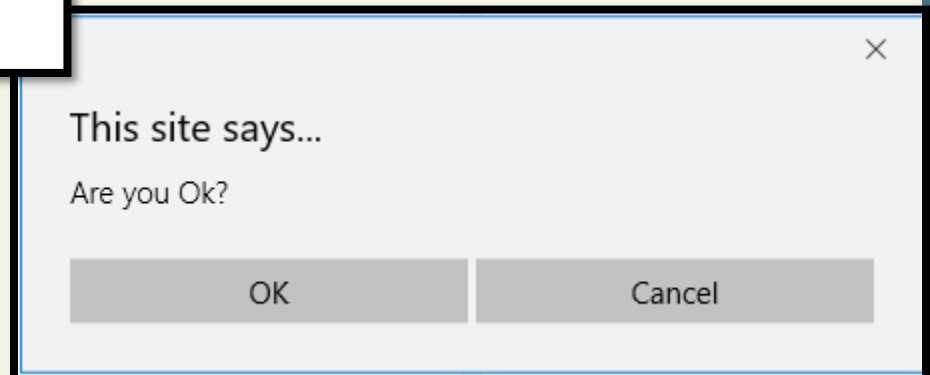
```
var person = prompt("Please enter your name", "Harry Potter");
```



window.confirm()

- We can also get user input by using the confirm method of the window object:
`window.confirm(question);`
- The method takes one arguments: the question to ask the user. Just like `window.prompt()`, we can assign its return value to a variable. The data type, however, is **boolean, not string!**

```
<script>  
window.confirm("Are you Ok?");  
</script>
```



window.confirm()

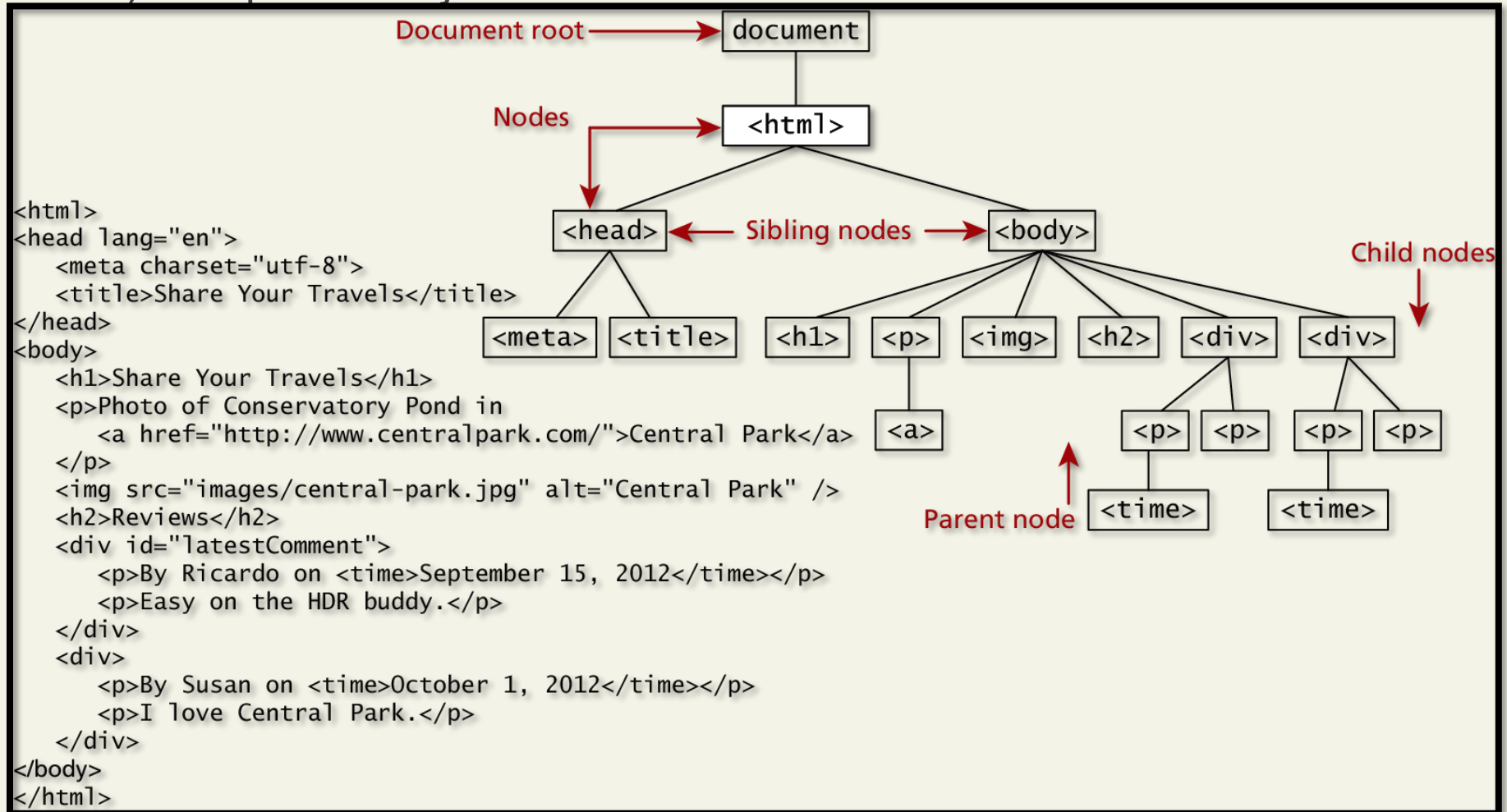
- When we use window.confirm(), the user will see a dialogue box with a question and two buttons: an OK button and a Cancel button.
- If the user clicks OK, the return value is true.
- If the user clicks Cancel, the return value is false.
- Confirm example

Section 6 of 8

THE DOCUMENT OBJECT MODEL (DOM)

The DOM

The **tree structure of the (HTML)** is formally called the **DOM Tree** with the root, or top most object called the **Document Root**.



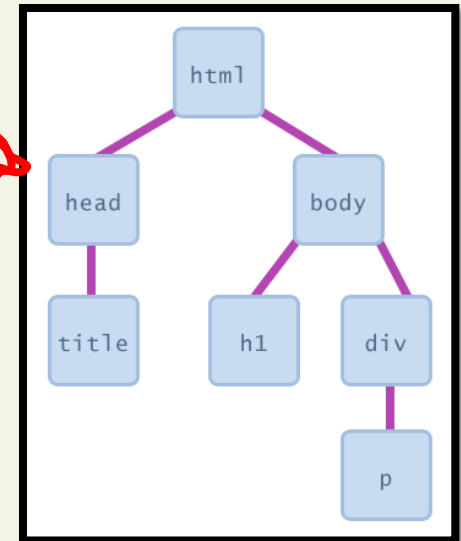
DOM Nodes

Document Object Model

In the DOM, each element within the HTML document is called a **node**. If the DOM is a **tree**, then each node is an individual branch.

There are:

- **element** nodes,
- **text** nodes, and
- **attribute** nodes



Document Object Model (DOM)

a set of JavaScript objects that represent each element on the page

most JS code manipulates elements on an HTML page

we can examine elements' state

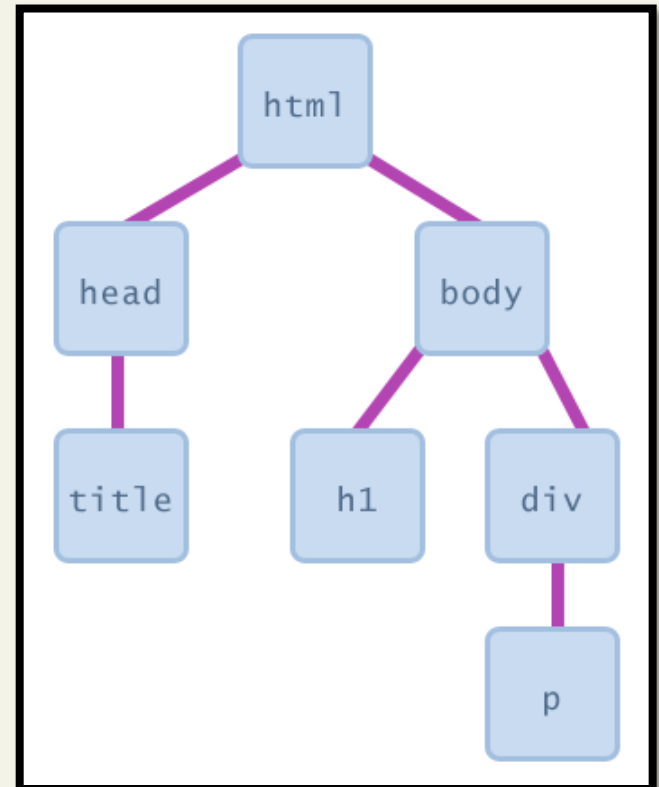
e.g. see whether a box is checked

we can change state

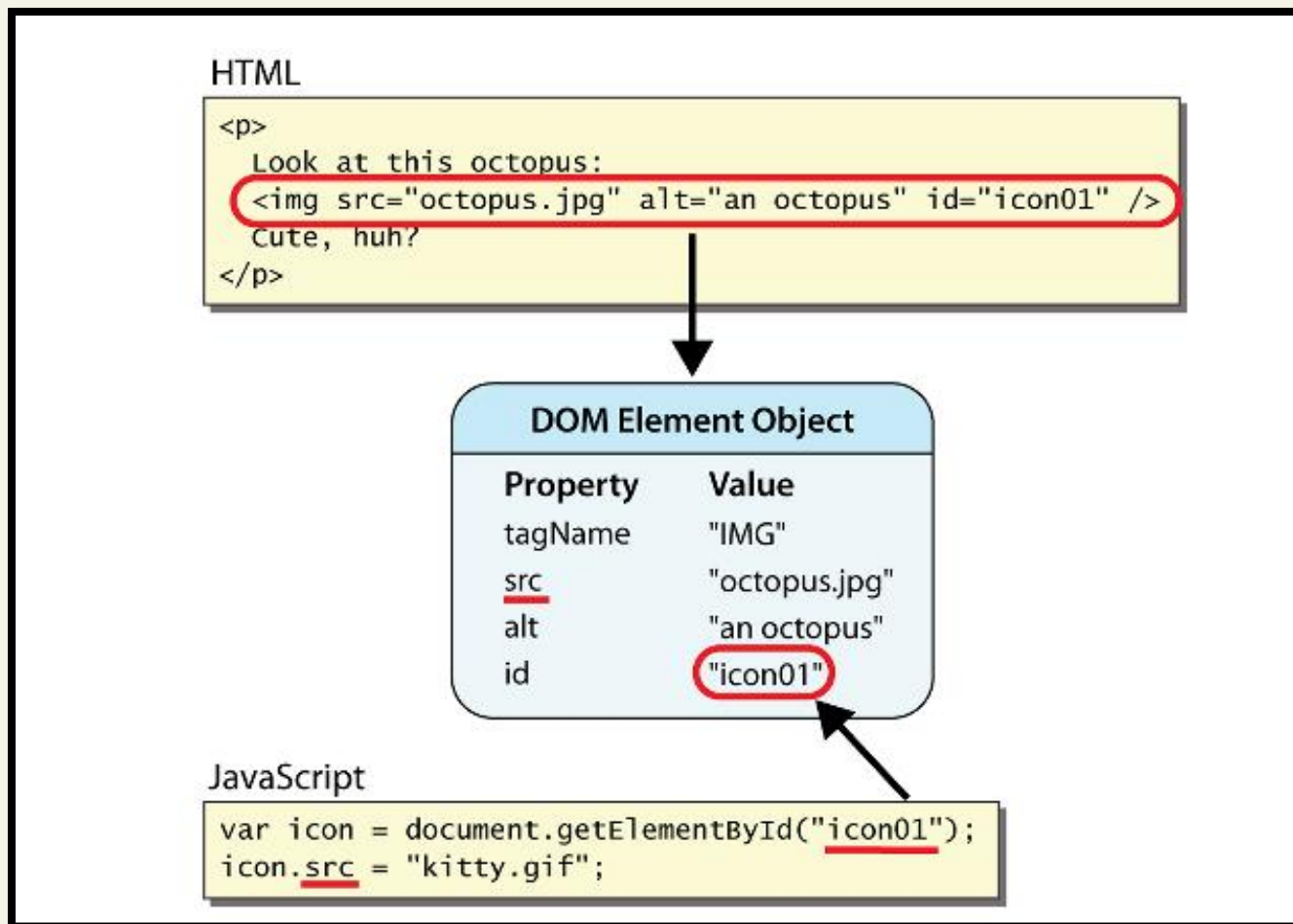
e.g. insert some new text into a div

we can change styles

e.g. make a paragraph red



DOM Nodes

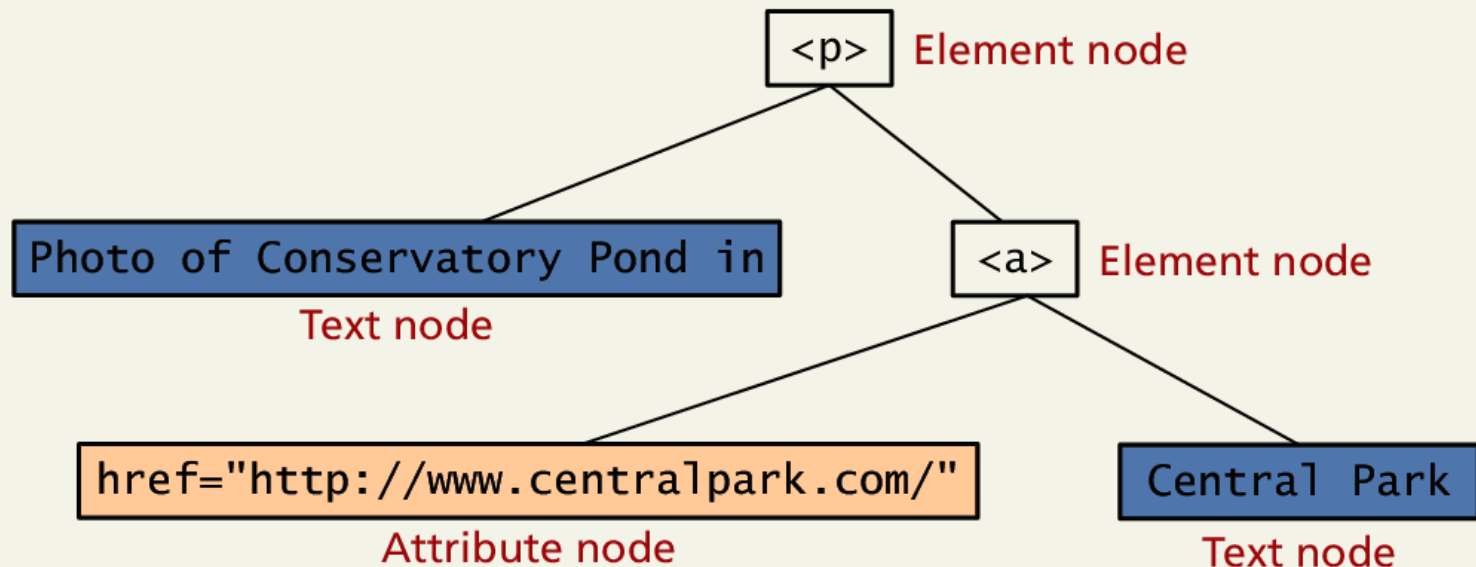


`document.getElementById`
returns the DOM object for an element with a given `id`

DOM Nodes

Element, text and attribute nodes

```
<p>Photo of Conservatory Pond in  
  <a href="http://www.centralpark.com/">Central Park</a>  
</p>
```



Accessing elements: document.getElementById

```
var name = document.getElementById("id");
```

```
<button onclick="changeText();" >Click me!</button>
```

```
<input id="output" type="text" value="replace me" />
```

```
function changeText() {  
  var textbox = document.getElementById("output");  
  textbox.value = "Hello, world!";  
}
```

Click me!

`document.getElementById` returns the DOM object for an element with a given id
can change the text in most form controls by setting the value property

More advanced example

```
<button onclick="swapText();">Click me!</button>  
<span id="output2">Hello</span>  
<input id="textbox2" type="text" value="Goodbye" />
```

```
function swapText() {  
  var span = document.getElementById("output2");  
  var textBox = document.getElementById("textbox2");  
  var temp = span.innerHTML;  
  span.innerHTML = textBox.value;  
  textBox.value = temp;  
}
```

Click me! Hello Goodbye

can change the text inside most elements by setting **the innerHTML** property

DOM Nodes

Essential Node Object properties

Property	Description
<code>attributes</code>	Collection of node attributes
<code>childNodes</code>	A <code>NodeList</code> of child nodes for this node
<code>firstChild</code>	First child node of this node.
<code>lastChild</code>	Last child of this node.
<code>nextSibling</code>	Next sibling node for this node.
<code>nodeName</code>	Name of the node
<code>nodeType</code>	Type of the node
<code>nodeValue</code>	Value of the node
<code>parentNode</code>	Parent node for this node.
<code>previousSibling</code>	Previous sibling node for this node.

```
<html>
<body><!-- This is a comment node! -->
```

```
<p>Click the button get info about the body element's child nodes.</p>
```

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

```
<p><strong>Note:</strong> Whitespace inside elements is considered as text,
and text
is considered as nodes. Comments are also considered as nodes.</p>
```

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

```
<script>
function myFunction() {
  var c = document.body.childNodes;
  var txt = "";
  var i;
  for (i = 0; i < c.length; i++) {
    txt = txt + c[i].nodeName + "<br>";
  }

  document.getElementById("demo").innerHTML = txt;
}
</script>
```

```
</body>
</html>
```

```
#comment
#text
P
#text
BUTTON
#text
P
#text
P
#text
SCRIPT
#text
```

[View on Browser](#)

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Example list:</p>
```

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

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

```
<p><strong>Note:</strong> Whitespace inside elements is considered as text,  
and text is considered as nodes.</p>
```

```
<p>If you add whitespace before the first LI element, the result will be  
"undefined".</p>
```

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

```
<script>
```

```
function myFunction() {
```

```
    var list = document.getElementById("myList").firstChild.innerHTML;
```

```
    document.getElementById("demo").innerHTML = list;
```

```
}
```

```
</script>
```

Coffee

```
</body>
```

```
</html>
```

[View on Browser](#)

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Example list:</p>
```

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

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

Note: Whitespace inside elements is considered as text, and text is considered as nodes.

If you add whitespace before the first LI element, the result will be "undefined".

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

```
<script>
```

```
function myFunction() {
```

```
    var list = document.getElementById("myList").firstChild.innerHTML;
```

```
    document.getElementById("demo").innerHTML = list;
```

```
}
```

```
</script>
```

undefined

```
</body>
```

```
</html>
```

[View on Browser](#)

Document Object

One root to ground them all

The **DOM document object** is the root JavaScript object representing the entire HTML document.

It contains some properties and methods that we will use extensively in our development and is globally accessible as **document**.

// specify the doctype, for example html

```
var a = document.doctype.name;
```

// specify the page encoding, for example ISO-8859-1, or WINDOWS-1256

```
var b = document.inputEncoding;
```

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

```
<script>
function myFunction() {
    var a = document.doctype.name;

    document.getElementById("demo").innerHTML = a;
}
</script>
```

html

Document Object

Document Object Methods

Method	Description
<code>createAttribute()</code>	Creates an attribute node
<code>createElement()</code>	Creates an element node
<code>createTextNode()</code>	Create a text node
<code>getElementById(id)</code>	Returns the element node whose <code>id</code> attribute matches the passed <code>id</code> parameter.
<code>getElementsByTagName(name)</code>	Returns a nodeList of elements whose tag name matches the passed <code>name</code> parameter.

Accessing nodes

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

<p>Click the button to make a BUTTON element.</p>
<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
    var btn = document.createElement("BUTTON");
    document.body.appendChild(btn);
}
</script>

</body>
</html>
```

Click the button to make a BUTTON element.

Try it

Click the button to make a BUTTON element.

Try it

Accessing nodes

```
var btn = document.createElement("BUTTON");  
var t = document.createTextNode("CLICK ME");  
btn.appendChild(t);  
document.body.appendChild(btn);
```

```
// Create a <button> element  
// Create a text node  
// Append the text to <button>  
// Append <button> to <body>
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<p>Click the button to make a BUTTON element with text.</p>  
  
<button onclick="myFunction()">Try it</button>  
  
<script>  
function myFunction() {  
  var btn = document.createElement("BUTTON");  
  var t = document.createTextNode("CLICK ME");  
  btn.appendChild(t);  
  document.body.appendChild(btn);  
}  
</script>  
  
</body>  
</html>
```

Click the button to make a BUTTON element with text.

Try it

Click the button to make a BUTTON element with text.

Try it

CLICK ME

New button

Accessing nodes

```
<html>
  <body>
    <h1> Comp334 </h1>

    <h2>Reviews</h2>
    <div id="test">
      <p>Very important example</p>
      <p>Just have a look</p>
    </div>
    <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Palestine.</p>
    </div>

    <script>
      var msg1 = document.getElementById("test"):

      alert(msg1.innerHTML);
    </script>
  </body>
</html>
```

Comp334

Reviews

Very important example

Just have a look

By Susan on October 1, 2012

I love Palestine.

This site says...

<p>Very important example</p>

<p>Just have a look</p>

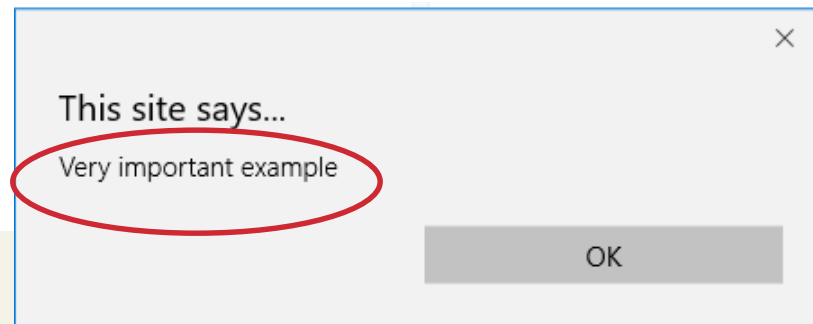
OK

Accessing nodes

```
<html>
  <body>
    <h1> Comp334 </h1>

    <h2>Reviews</h2>
    <div id="test"><p>Very important example</p><p>Just have a look</p>
    </div>
    <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Palestine.</p>
    </div>

    <script>
      var msg1 = document.getElementById("test");
      alert(msg1.firstChild.innerHTML);
    </script>
  </body>
</html>
```



Comp334

Reviews

Very important example

Just have a look

By Susan on October 1, 2012

I love Palestine.

Accessing nodes

```
<html>
  <body>
    <h1> Comp334 </h1>

    <h2>Reviews</h2>
    <div id="test">
      <p>Very important example</p>
      <p>Just have a look</p>
    </div>
    <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Palestine.</p>
    </div>

    <script>
      var msg1 = document.getElementById("test");
      alert(msg1.firstChild.innerHTML);
    </script>
  </body>
</html>
```

space

Comp334

Reviews

Very important example

Just have a look

By Susan on October 1, 2012

I love Palestine.

This site says...

undefined

OK

Accessing nodes

getElementById(), getElementsByTagName()

```
var abc = document.getElementById("latestComment");
```

```
<body>
  <h1>Reviews</h1>
  <div id="latestComment">
    <p>By Ricardo on <time>September 15, 2012</time></p>
    <p>Easy on the HDR buddy.</p>
  </div>
  <hr/>
  <div>
    <p>By Susan on <time>October 1, 2012</time></p>
    <p>I love Central Park.</p>
  </div>
  <hr/>
</body>
```

```
var list = document.getElementsByTagName("div");
```

Element node Object

The type of object returned by the method `document.getElementById()` described in the previous section is an **element node object**.

This represents an HTML element in the hierarchy, contained between the opening `<>` and closing `</>` tags for this element.

- **can itself contain more elements**

Element node Object

Essential Element Node Properties

Property	Description
className	The current value for the class attribute of this HTML element.
id	The current value for the id of this element.
innerHTML	Represents all the things inside of the tags. This can be read or written to and is the primary way which we update particular div's using JS.
style	The style attribute of an element. We can read and modify this property.
tagName	The tag name for the element.

Modifying a DOM element

The `document.write()` method is used to create output to the HTML page from JavaScript.

```
<script>  
document.write("Hello World!");  
</script>
```

```
document.write("<h1>Hello World</h1>");
```

The modern JavaScript programmer will want to write to the HTML page, but in **a particular location**

Using the DOM document and HTML DOM element objects, we can do exactly that using the **innerHTML property**

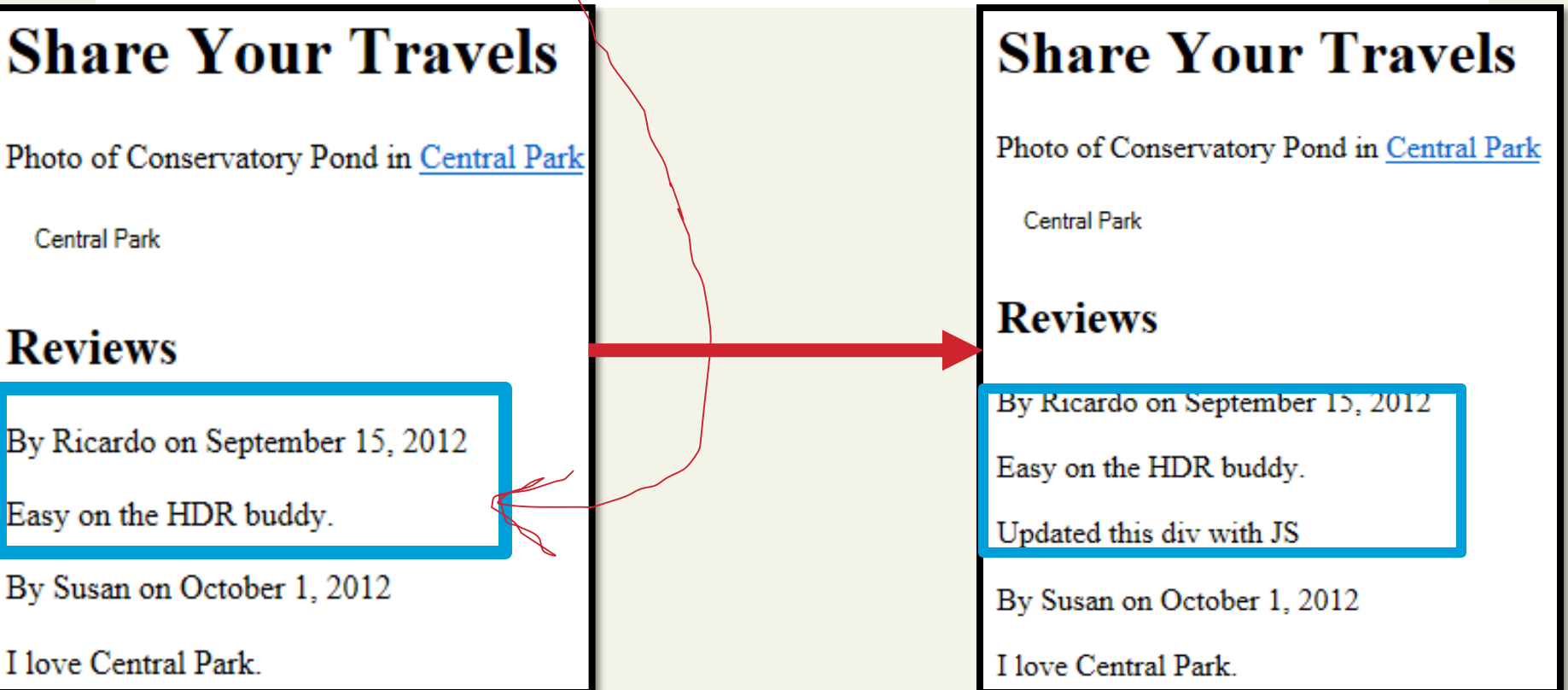
```
var latest = document.getElementById("latestComment");  
var oldMessage = latest.innerHTML;  
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

LISTING 6.8 Changing the HTML using innerHTML

Modifying a DOM element

```
var latest = document.getElementById("latestComment");  
var oldMessage = latest.innerHTML;  
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

LISTING 6.8 Changing the HTML using innerHTML



Modifying a DOM element

```
var latest = document.getElementById("latestComment");  
var oldMessage = latest.innerHTML;  
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

LISTING 6.8 Changing the HTML using innerHTML

Result



```
<div id="latestComment">  
  <p>By Ricardo on <time>September 15, 2012</time></p>  
  <p>Easy on the HDR buddy.</p>  
  <p>Updated this div with JS</p>  
</div>
```

Modifying a DOM element

```
<html>
  <body>
    <h1> Comp334 </h1>

    <h2>Reviews</h2>
    <div id="test"><p>Very important example</p><p>Just have a look</p>
    </div>
    <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Palestine.</p>
    </div>

    <script>
      var msg1 = document.getElementById("test");
      msg1.removeChild(msg1.firstChild);
    </script>
  </body>
</html>
```

Comp334

Reviews

Very important example

Just have a look

By Susan on October 1, 2012

I love Palestine.

Before

Comp334

Reviews

Just have a look

By Susan on October 1, 2012

I love Palestine.

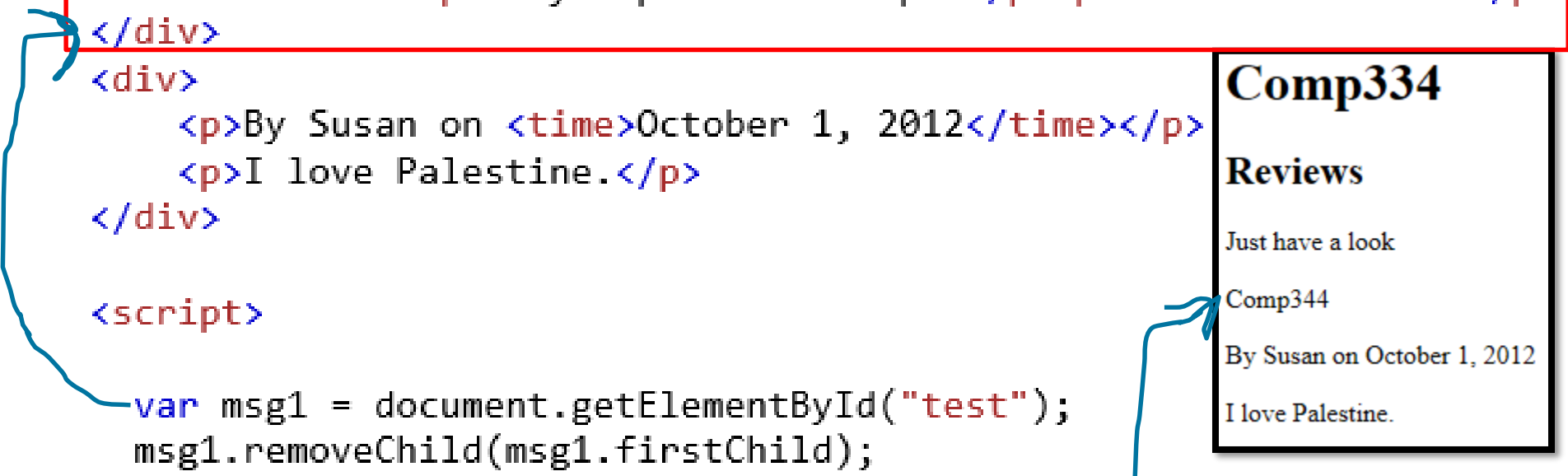
After

Modifying a DOM element

```
<html>
  <body>
    <h1> Comp334 </h1>

    <h2>Reviews</h2>
    <div id="test"><p>Very important example</p><p>Just have a look</p>
    </div>
    <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Palestine.</p>
    </div>

    <script>
      var msg1 = document.getElementById("test");
      msg1.removeChild(msg1.firstChild);
      msg1.appendChild( document.createTextNode("Comp344"));
    </script>
  </body>
</html>
```



Comp334

Reviews

Just have a look

Comp344

By Susan on October 1, 2012

I love Palestine.

Modifying a DOM element

More verbosely, and validated

Although the innerHTML technique works well (and is very fast), there is a more verbose technique available to us that builds output using the DOM.

DOM functions `createTextNode()`, `removeChild()`, and `appendChild()` allow us to modify an element in a more rigorous way

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
var newMessage = oldMessage + "<p>Updated this div with JS</p>";
latest.removeChild(latest.firstChild);
latest.appendChild(document.createTextNode(newMessage));
```

LISTING 6.9 Changing the HTML using `createTextNode()` and `appendChild()`

Changing an element's style

We can add or remove any style using the **style** or **className** property of the Element node.

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

Example:

```
document.getElementById("p2").style.color = "blue";
```


Changing an element's style

We can add or remove any style using the **style** or **className** property of the Element node.

Its usage is shown below to change a node's background color and add a three-pixel border.

```
var commentTag = document.getElementById("specificTag");  
  
commentTag.style.backgroundColor = "#FFFF00";  
  
commentTag.style.borderWidth="3px";
```

Example

Changing an element's style

With class

The `className` property is normally a better choice, because it allows the styles to be created outside the code, and thus be better accessible to designers.

```
var commentTag = document.getElementById("specificTag");  
commentTag.className = "someClassName";
```

HTML5 introduces the `classList` element, which allows you to add, remove, or toggle a CSS class on an element.

```
label.classList.addClass("someClassName");
```

Example (styling an element with a specific class)

Example (adding a class to an element)

Example (selecting elements by className)

Changing an element's style

```
<head>
<style>
.mystyle {
  width: 300px;
  height: 50px;
  background-color: coral;
  font-size: 25px;
}
.newStyle{color: white;}
</style>
</head>
<body>
<p>Click the button to add the "mystyle" class to DIV.</p>

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

<div id="myDIV">
I am a DIV element
</div>

<script>
function myFunction() {
  document.getElementById("myDIV").classList.add("mystyle");
  document.getElementById("myDIV").classList.add("newStyle");
}
</script>
</body>
</html>
```

Click the button to add the "mystyle" class to DIV.

Try it

I am a DIV element

Click the button to add the "mystyle" class to DIV.

Try it

I am a DIV element

Common DOM styling errors

- many students forget to write **.style** when setting styles

```
var clickMe = document.getElementById("clickme");  
clickMe.color = "red"; //wrong  
clickMe.style.color = "red"; //true
```

- style properties are capitalized** like This, not like-this

```
clickMe.style.font-size = "14pt"; //wrong  
clickMe.style.fontSize = "14pt"; //true
```

- style properties must be **set as strings**, often with units at the end

```
clickMe.style.width = 200; //wrong  
clickMe.style.width = "200px";  
clickMe.style.padding = "0.5em";
```

- write exactly the value you would have written in the CSS, but in quotes

More Properties

Some Specific HTML DOM Element Properties for Certain Tag Types

Property	Description	Tags
href	The href attribute used in a tags to specify a URL to link to.	a
name	The name property is a bookmark to identify this tag. Unlike id which is available to all tags, name is limited to certain form related tags.	a, input, textarea, form
src	Links to an external URL that should be loaded into the page (as opposed to href which is a link to follow when clicked)	img, input, iframe, script
value	The value is related tot he value attribute of input tags. Often the value of an input field is user defined, and we use value to get that user input.	Input, textarea, submit

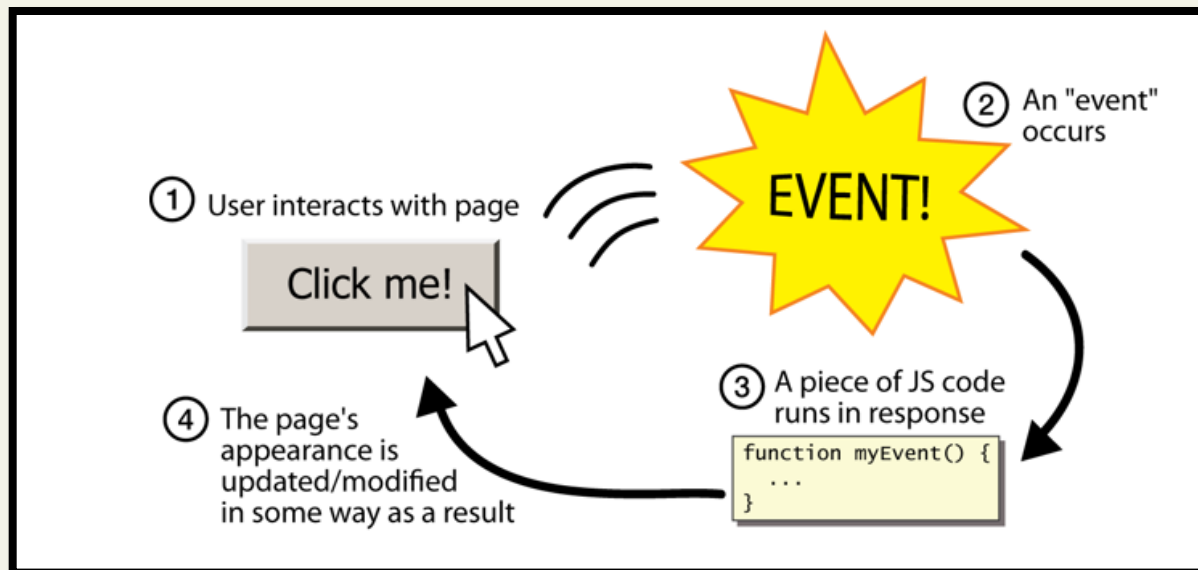
Section 7 of 8

JAVASCRIPT EVENTS

JavaScript Events

A JavaScript **event** is an action that can be detected by JavaScript.

We say then that *an event is triggered* and then it can *be caught by JavaScript functions*, which then do something in response.



JavaScript Events

A brave new world

```
<button onclick="okayClick();" >OK</button> HTML  
// called when OK button is clicked  
function okayClick() {  
  alert(" Comp334 ");  
} JS
```

OK *output*

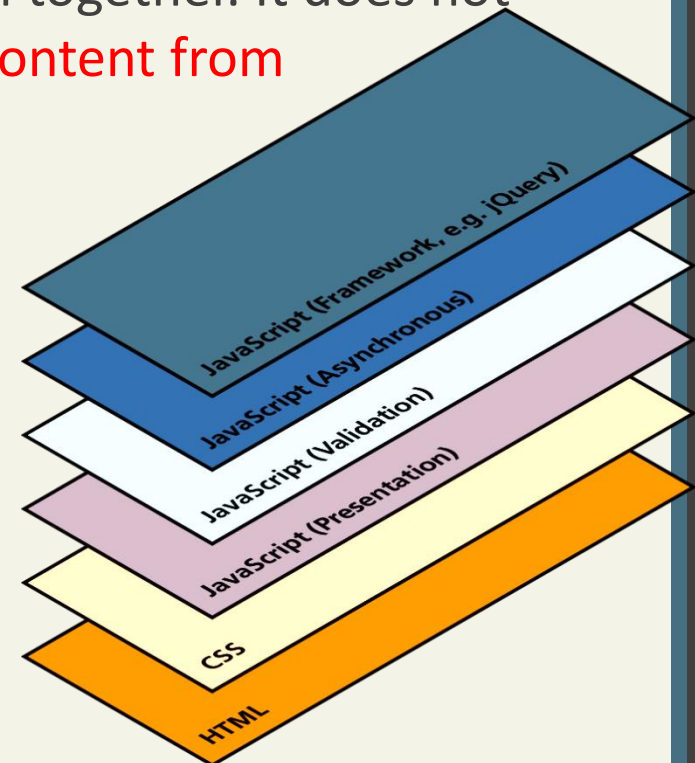
- This is bad style (HTML is cluttered with JS code)
- Goal: remove all JavaScript code from the HTML body

Inline Event Handler Approach

For example, if you wanted an alert to pop-up when clicking a <div> you might program:

```
<div id="example1" onclick="alert('hello')">Click for pop-up</div>
```

The problem with this type of programming is that the HTML markup and the corresponding JavaScript logic are woven together. It does not make use of layers; that is, it **does not separate content from behavior**.



Listener Approach

Two ways to set up listeners

```
var greetingBox = document.getElementById('example1');  
greetingBox.onclick = alert('Good Morning');
```

LISTING 6.10 The “old” style of registering a listener.

```
var greetingBox = document.getElementById('example1');  
greetingBox.addEventListener('click', alert('Good Morning'));  
greetingBox.addEventListener('mouseout', alert('Goodbye'));  
  
// IE 8  
greetingBox.attachEvent('click', alert('Good Morning'));
```

LISTING 6.11 The “new” DOM2 approach to registering listeners.

Listener Approach

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

<p>This example uses the addEventListener() method to attach a click event to a
button.</p>

<button id="myBtn">Try it</button>

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

```
<script>
document.getElementById("myBtn").addEventListener("click", displayDate);

function displayDate() {
    document.getElementById("demo").innerHTML = Date();
}
</script>
```

```
</body>
</html>
```

This example uses the addEventListener() method to attach a click event to a button.

Try it

This example uses the addEventListener() method to attach a click event to a button.

Try it

Sun Apr 22 2018 13:15:01 GMT+0300 (Jerusalem Daylight Time)

Before

After

Listener Approach

Using functions

```
function displayTheDate() {  
    var d = new Date();  
    alert ("You clicked this on "+ d.toString());  
}  
var element = document.getElementById('example1');  
element.onclick = displayTheDate;  
  
// or using the other approach  
element.addEventListener('click', displayTheDate);
```

LISTING 6.12 Listening to an event with a function

Listener Approach

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
```

Show

```
Moused over!
Moused out!
Moused over!
Clicked!
Clicked!
Clicked!
Clicked!
Moused out!
Moused over!
Moused out!
Moused over!
Moused out!
```

```
<script>
document.addEventListener("mouseover", myFunction);
document.addEventListener("click", mySecondFunction);
document.addEventListener("mouseout", myThirdFunction);
```

```
function myFunction() {
    document.getElementById("demo").innerHTML += "Moused over!<br>"
}
```

```
function mySecondFunction() {
    document.getElementById("demo").innerHTML += "Clicked!<br>"
}
```

```
function myThirdFunction() {
    document.getElementById("demo").innerHTML += "Moused out!<br>"
}
```

```
</script>
```

```
</body>
```

```
</html>
```

You can add events of different types to the document.
This example demonstrates how to add many events to the document

Listener Approach

Anonymous functions

An alternative to that shown in Listing 6.12 is to use an **anonymous function (that is, one without a name)**, as shown in Listing 6.13.

```
var element = document.getElementById('example1');
element.onclick = function() {
    var d = new Date();
    alert ("You clicked this on " + d.toString());
};
```

LISTING 6.13 Listening to an event with an anonymous function

```
document.onclick=function() {
    document.getElementById("demo").innerHTML += "Clicked!<br>"
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Click anywhere in the document.</p>
```

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

```
<script>
```

```
document.addEventListener("click", function(){  
    document.getElementById("demo").innerHTML = "Hello World!";  
});
```

```
</script>
```

```
</body>
```

```
</html>
```

Click anywhere in the document.

Hello World!

After

Click anywhere in the document.

Before

Event Object

No matter which type of event we encounter, they are all **DOM event objects** and the event handlers associated with them can access and manipulate them. Typically we see **the events passed to the function handler as a parameter named *e***.

```
function someHandler(e) {  
    // e is the event that triggered this handler.  
}
```


Event Types

There are several classes of event, with several types of event within each class specified by the W3C:

- mouse events
- keyboard events
- form events
- frame events

Mouse events

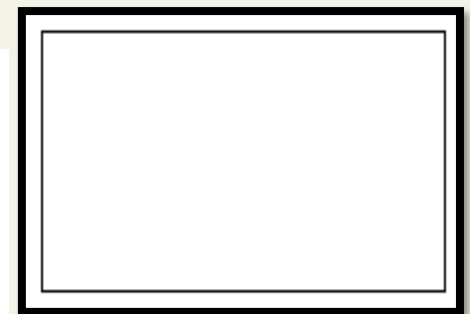
Event	Description
onclick	The mouse was clicked on an element
ondblclick	The mouse was double clicked on an element
onmousedown	The mouse was pressed down over an element
onmouseup	The mouse was released over an element
onmouseover	The mouse was moved (not clicked) over an element
onmouseout	The mouse was moved off of an element
onmousemove	The mouse was moved while over an element

Mouse events

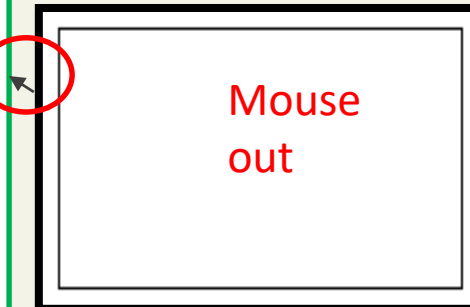
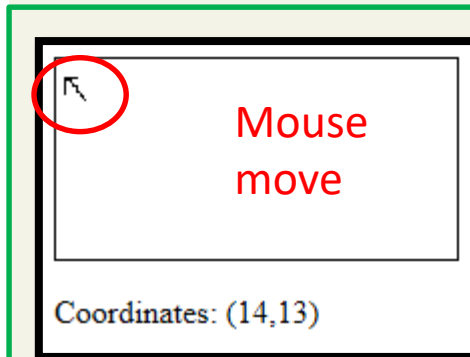
```
<html>
<head>
<style>
div {
  width: 200px;
  height: 100px;
  border: 1px solid black;
}
</style>
</head>
<body>
<div onmousemove="myFunction(event)" onmouseout="clearCoor()"></div>
<p id="demo"></p>
<script>
function myFunction(e) {
  var x = e.clientX;
  var y = e.clientY;
  var coor = "Coordinates: (" + x + ", " + y + ")";
  document.getElementById("demo").innerHTML = coor;
}

function clearCoor() {
  document.getElementById("demo").innerHTML = "";
}
</script>
</body>
</html>
```

[Show](#)



Before



After

Keyboard events

Event	Description
onkeydown	The user is pressing a key (this happens first)
onkeypress	The user presses a key (this happens after onkeydown)
onkeyup	The user releases a key that was down (this happens last)

Keyboard events

Example

```
<input type="text" id="keyExample">
```

The input box above, for example, could be listened to and each key pressed echoed back to the user as an alert as shown in Listing 6.15.

```
document.getElementById("keyExample").onkeydown = function
myFunction(e){
    var keyPressed=e.keyCode;           //get the raw key code
    var character=String.fromCharCode(keyPressed); //convert to string
    alert("Key " + character + " was pressed");
}
```

LISTING 6.15 Listener that hears and alerts keypresses

Example

Get the Unicode value of the pressed keyboard key:

```
var x = event.keyCode;
```

Example

Convert a Unicode number into a character:

```
var res = String.fromCharCode(65);
```

Keyboard events

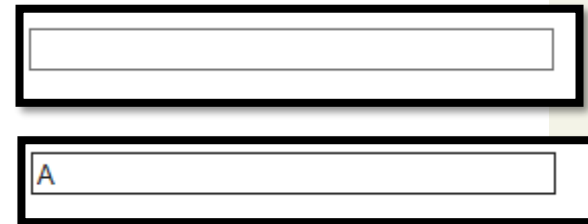
```
<!DOCTYPE html>
<html>
<body>

<p>A function is triggered when the user is pressing a key in the input
field.</p>

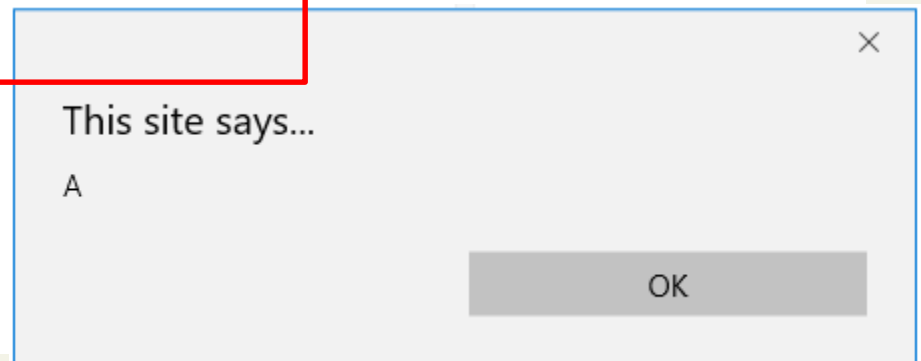
<input type="text" id="txt">

<script>
var obj=document.getElementById("txt");
obj.onkeydown=function(e){
  var keyPress=e.keyCode;
  var character=String.fromCharCode(keyPress);
  alert(character);
}
</script>

</body>
</html>
```



Two text input fields are shown. The top field is empty, and the bottom field contains the letter 'A'. A red arrow points from the 'id="txt"' attribute in the code to the top input field.



Form Events

Event	Description
onblur	A form element has lost focus (that is, control has moved to a different element, perhaps due to a click or Tab key press).
onchange	Some <input>, <textarea> or <select> field had their value change . This could mean the user typed something, or selected a new choice.
onfocus	Complementing the onblur event, this is triggered when an element gets focus (the user clicks in the field or tabs to it)
onreset	HTML forms have the ability to be reset . This event is triggered when that happens.
onselect	When the users selects some text. This is often used to try and prevent copy/paste.
onsubmit	When the form is submitted this event is triggered . We can do some pre-validation when the user submits the form in JavaScript before sending the data on to the server.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>This example uses the addEventListener() method to attach a "focus" event to an input element.</p>
```

```
Enter your name: <input type="text" id="fname">
```

```
<script>
```

```
document.getElementById("fname").addEventListener("focus", myFunction);
```

```
function myFunction() {  
    document.getElementById("fname").style.backgroundColor = "red";  
}
```

```
</script>
```

```
</body>
```

```
</html>
```

This example uses the addEventListener() method to attach a "focus" event to an input element.

Enter your name:

Before

This example uses the addEventListener() method to attach a "focus" event to an input element.

Enter your name:

After


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

<p>This example uses the addEventListener() method to attach a "change" event
to an input element.</p>

Enter your name: <input type="text" id="fname">

<p>When you leave the input field, a function is triggered which transforms
the input text to upper case.</p>

<script>
document.getElementById("fname").addEventListener("change", myFunction);

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

</body>
</html>
```

This example uses the addEventListener() method to attach a "change" event to an input element.

Enter your name:

When you leave the input field, a function is triggered which transforms the input text to upper case.

Before

This example uses the addEventListener() method to attach a "change" event to an input element.

Enter your name:

When you leave the input field, a function is triggered which transforms the input text to upper case.

After

Frame Events

Frame events are the events related to the browser frame that contains your web page.

The most important event is the **onload** event, which tells us an object is loaded and therefore ready to work with. If the code attempts to set up a listener on this not-yet-loaded <div>, then an error will be triggered.

```
window.onload= function(){  
  
    //all JavaScript initialization here.  
  
}
```

Frame Events

Table of frame events

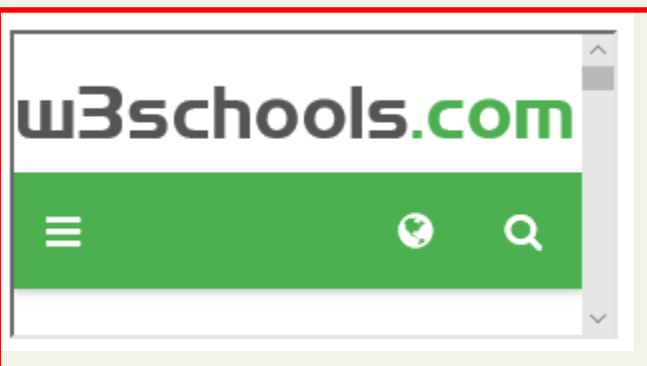
Event	Description
onabort	An object was stopped from loading
onerror	An object or image did not properly load
onload	When a document or object has been loaded
onresize	The document view was resized
onscroll	The document view was scrolled
onunload	The document has unloaded

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

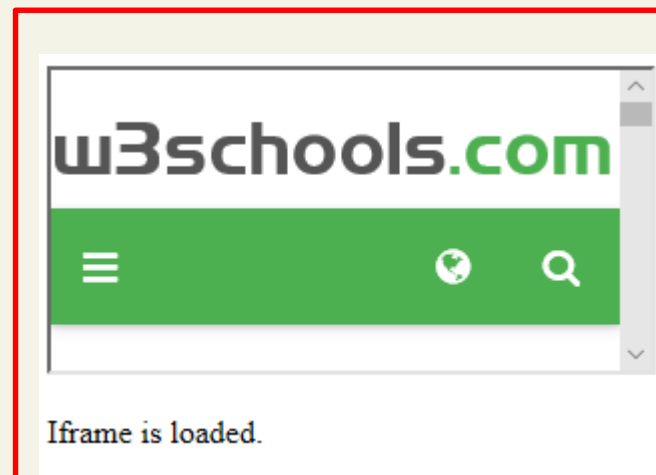
```
<iframe id="myFrame" src="/default.asp"></iframe>
```

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

```
<script>  
document.getElementById("myFrame").addEventListener("load", myFunction);  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Iframe is loaded.";  
}  
</script>  
</body>  
</html>
```



Before



After

Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```

```

A function is triggered if an error occurs when loading the image. The function shows an alert box with a text. In this example we refer to an image that does not exist, therefore the onerror event occurs.

```
<script>
```

```
function myFunction() {  
    alert('The image could not be loaded.');
```

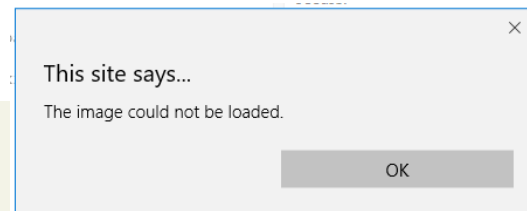
```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

A function is triggered if an error occurs when loading the image. The function shows an alert box with a text. In this example we refer to an image that does not exist, therefore the onerror event occurs.



After

Section 8 of 8

FORMS

Validating Forms

You mean pre-validating right?

Writing code to **prevalidate forms** on the client side will reduce the number of incorrect submissions, thereby reducing server load.

There are a number of common validation activities including **email validation, number validation, and data validation.**

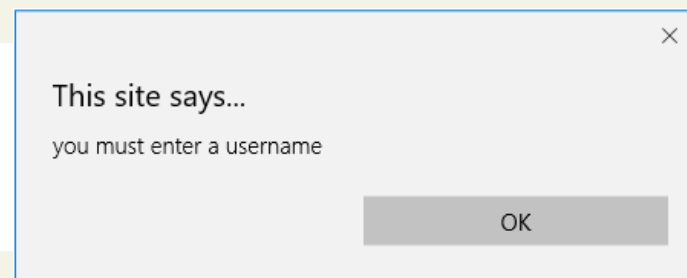
Validating Forms

```
<body>
  <h1>Listing 6.18 HTML Page</h1>
  <form action='login.php' method='post' id='loginForm'>
    <input type='text' name='username' id='username' />
    <input type='password' name='password' id='password' />
    <input type='submit'></input>
  </form>

  <script type="text/JavaScript" src="Listing6.18.js">
  </script>
</body>
```

```
document.getElementById("loginForm").onsubmit = function(e){
  var fieldValue=document.getElementById("username").value;
  if(fieldValue==null || fieldValue== ""){
    // the field was empty. Stop form submission
    e.preventDefault();
    // Now tell the user something went wrong
    alert("you must enter a username");
  }
}
```

Listing 6.18 HTML Page



[show](#)

Validating Forms

Empty field

If you want to ensure a checkbox is ticked, use code like that below.

```
var inputField=document.getElementById("license");  
  
if (inputField.type=="checkbox"){  
    if (inputField.checked)  
        //Now we know the box is checked  
}
```

Validating Forms

Number Validation

```
function isNumeric(n) {  
    return !isNaN(parseFloat(n)) && isFinite(n);  
}
```

LISTING 6.19 A function to test for a numeric value

The `isNaN()` function determines whether a value is an illegal number (Not-a-Number).

The `isFinite()` function returns `false` if the value is `+infinity`, `-infinity`, or `NaN`, otherwise it returns `true`.

```
var a = isFinite(123); //true  
var d = isFinite(3/0) ;//false  
var g = isFinite("2005/12/12"); //false
```

```
isNaN(123) //false  
isNaN(-1.23) //false  
isNaN(5-2) //false  
isNaN(0) //false  
isNaN('123') //false  
isNaN('Hello') //true  
isNaN('2005/12/12') //true  
isNaN('') //false  
isNaN(true) //false  
isNaN(undefined) //true  
isNaN('NaN') //true  
isNaN(NaN) //true  
isNaN(0 / 0) //true
```

[Example](#)

[Example](#)

Validating Forms

More to come in Chapter 12

Form validation uses regular expressions, covered in Chapter 12

Submitting Forms

Submitting a form using JavaScript requires having a node variable for the form element. Once the variable, say, `formExample` is acquired, one can simply call the `submit()` method:

```
var formExample = document.getElementById("loginForm");
```

```
formExample.submit();
```

This is often done in conjunction with calling **`preventDefault()`** on the `onsubmit` event.

Event Object

Event Object

Several Options

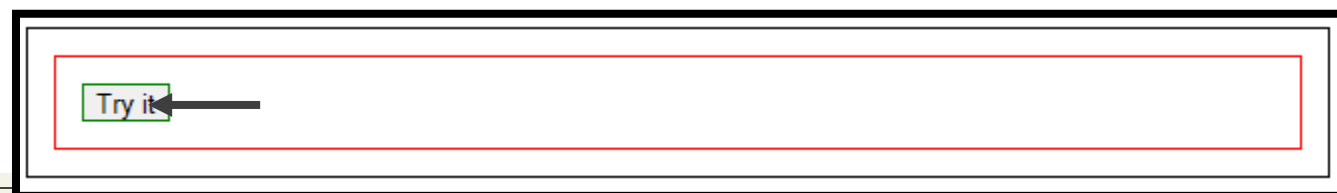
- **Bubbles.** If an event's bubbles property is set to true then there must be an event handler in place to handle the event or it will bubble up to its parent and trigger an event handler there.
- **Cancelable.** The Cancelable property is also a Boolean value that indicates **whether or not the event can be cancelled.**
- **preventDefault.** A cancelable default action for an event can be stopped using the preventDefault() method in the next slide

```
<!DOCTYPE html>
<html>
<body>
  <div id="div1" style="border: 1px solid black; padding: 13px;">
    <div id="div2" style="border: 1px solid red;padding: 13px">
      <button id="btn" style="border: 1px solid green;">Try it</button>
    </div>
  </div>
<p id="demo"></p>
<script>
document.getElementById("btn").addEventListener("click", function(event) {

  document.getElementById("demo").innerHTML += "Button ";
})
document.getElementById("div2").addEventListener("click", function(event) {

  document.getElementById("demo").innerHTML += "div2 ";
})
document.getElementById("div1").addEventListener("click", function(event) {

  document.getElementById("demo").innerHTML += "div1 ";
})
</script>
</body>
</html>
```



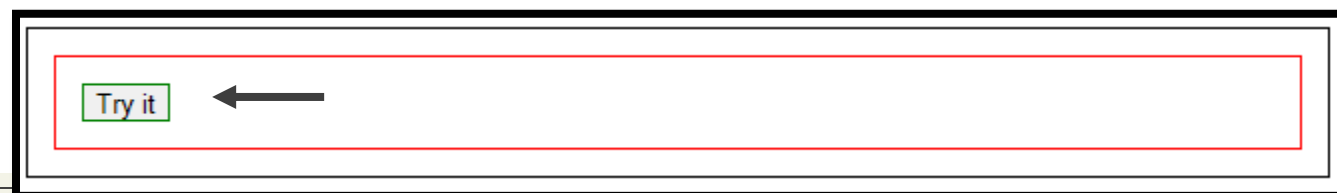
Button div2 div1

```
<!DOCTYPE html>
<html>
<body>
  <div id="div1" style="border: 1px solid black; padding: 13px;">
    <div id="div2" style="border: 1px solid red;padding: 13px">
      <button id="btn" style="border: 1px solid green;">Try it</button>
    </div>
  </div>
<p id="demo"></p>
<script>
document.getElementById("btn").addEventListener("click", function(event) {

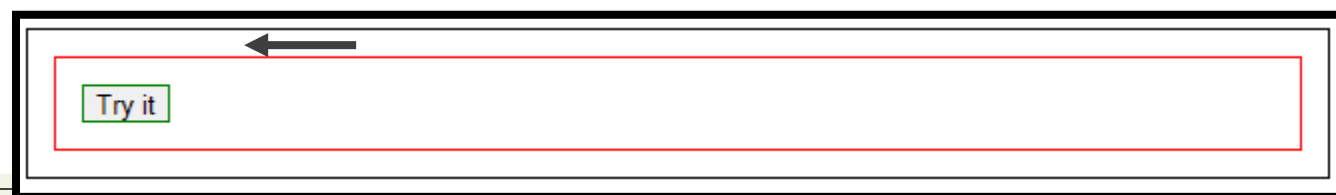
  document.getElementById("demo").innerHTML += "Button ";
})
document.getElementById("div2").addEventListener("click", function(event) {

  document.getElementById("demo").innerHTML += "div2 ";
})
document.getElementById("div1").addEventListener("click", function(event) {

  document.getElementById("demo").innerHTML += "div1 ";
})
</script>
</body>
</html>
```




```
<!DOCTYPE html>
<html>
<body>
  <div id="div1" style="border: 1px solid black; padding: 13px;">
    <div id="div2" style="border: 1px solid red;padding: 13px">
      <button id="btn" style="border: 1px solid green;">Try it</button>
    </div>
  </div>
<p id="demo"></p>
<script>
document.getElementById("btn").addEventListener("click", function(event) {
  document.getElementById("demo").innerHTML += "Button ";
})
document.getElementById("div2").addEventListener("click", function(event) {
  document.getElementById("demo").innerHTML += "div2 ";
})
document.getElementById("div1").addEventListener("click", function(event) {
  document.getElementById("demo").innerHTML += "div1 ";
})
</script>
</body>
</html>
```



div1

```
<html>
<body>
  <div id="div1" style="border: 1px solid black; padding: 13px;">
    <div id="div2" style="border: 1px solid red;padding: 13px">
      <button id="btn" style="border: 1px solid green;">Try it</button>
    </div>
  </div>
<p id="demo"></p>
<script>
document.getElementById("btn").addEventListener("mouseover", function(event) {
    document.getElementById("demo").innerHTML += "Button ";
})
document.getElementById("div2").addEventListener("click", function(event) {
    document.getElementById("demo").innerHTML += "div2 ";
})
document.getElementById("div1").addEventListener("click", function(event) {
    document.getElementById("demo").innerHTML += "div1 ";
})
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
  <div id="div1" style="border: 1px solid black; padding: 13px;">
    <div id="div2" style="border: 1px solid red;padding: 13px">
      <button id="btn" style="border: 1px solid green;">Try it</button>
    </div>
  </div>
<p id="demo"></p>
<script>
document.getElementById("btn").addEventListener("mouseover", function(event) {
    document.getElementById("demo").innerHTML += "Button ";
})
document.getElementById("div2").addEventListener("mouseover", function(event) {
    document.getElementById("demo").innerHTML += "div2 ";
})
document.getElementById("div1").addEventListener("click", function(event) {
    document.getElementById("demo").innerHTML += "div1 ";
})
</script>
</body>
</html>
```

div2 div2 Button div2 div2 div2 div2 Button div2 div2



Try it

```

<!DOCTYPE html>
<html>
<body>
  <div id="div1" style="border: 1px solid black; padding: 13px;">
    <div id="div2" style="border: 1px solid red;padding: 13px">
      <button id="btn" style="border: 1px solid green;">Try it</button>
    </div>
  </div>
<p id="demo"></p>
<script>
/*document.getElementById("btn").addEventListener("click", function(event) {
    document.getElementById("demo").innerHTML += "Button ";
})*
/*document.getElementById("div2").addEventListener("click", function(event) {
    document.getElementById("demo").innerHTML += "div2 ";
})*
document.getElementById("div1").addEventListener("click", function(event) {
    document.getElementById("demo").innerHTML += "div1 ";
})
</script>
</body>
</html>

```

div1 div1 div1

Try it

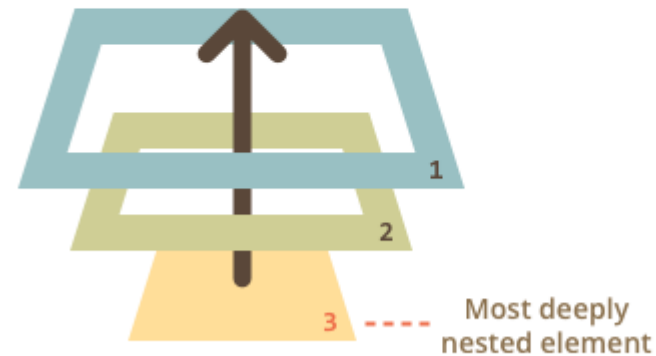
we have 3 nested elements FORM > DIV > P with a handler on each of them:

```
<style>
  body * {
    margin: 10px;
    border: 1px solid blue;
  }
</style>

<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```

A click on the inner `<p>` first runs `onclick`:

1. On that `<p>`.
2. Then on the outer `<div>`.
3. Then on the outer `<form>`.
4. And so on upwards till the `document` object.



So if we click on `<p>`, then we'll see 3 alerts: `p` → `div` → `form`.

FORM

DIV

P

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

<a id="ref" href="http://www.ritaj.birzeit.edu"> Ritaj</a>

<script>
var obj=document.getElementById("ref");
obj.addEventListener("click",function(event){

    event.preventDefault()

})

</script>
</body>
</html>
```

Ritaj

The default action that belongs to the event will not occur

Event Object

Prevent the default behaviour

```
function submitButtonClicked(e) {  
  if(e.cancelable){  
    e.preventDefault();  
  }  
}
```

LISTING 6.14 A sample event handler function that prevents the default event