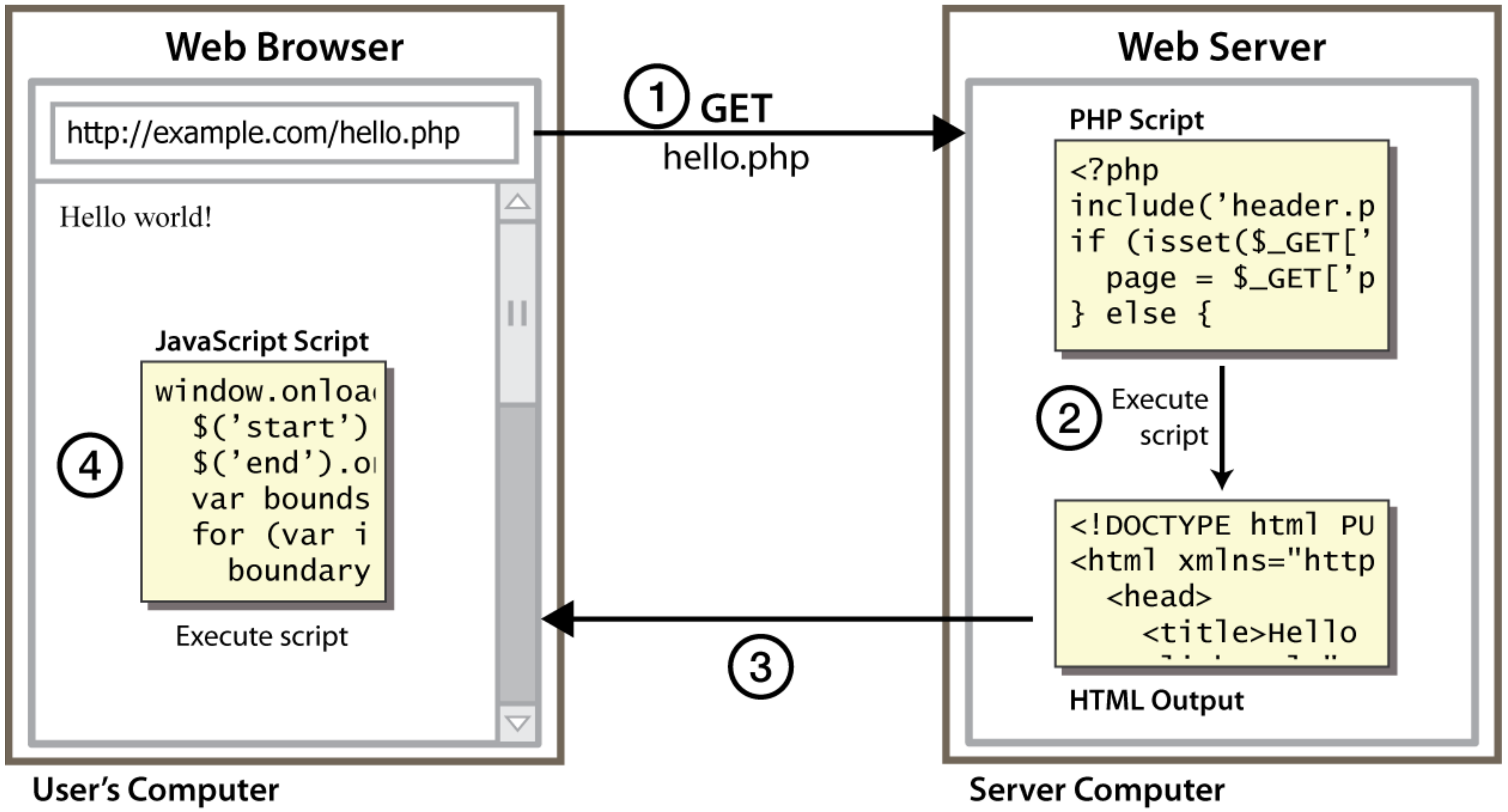
JavaScript

by:

Dr. Vandana Bhatia

Contents

Client Side Scripting



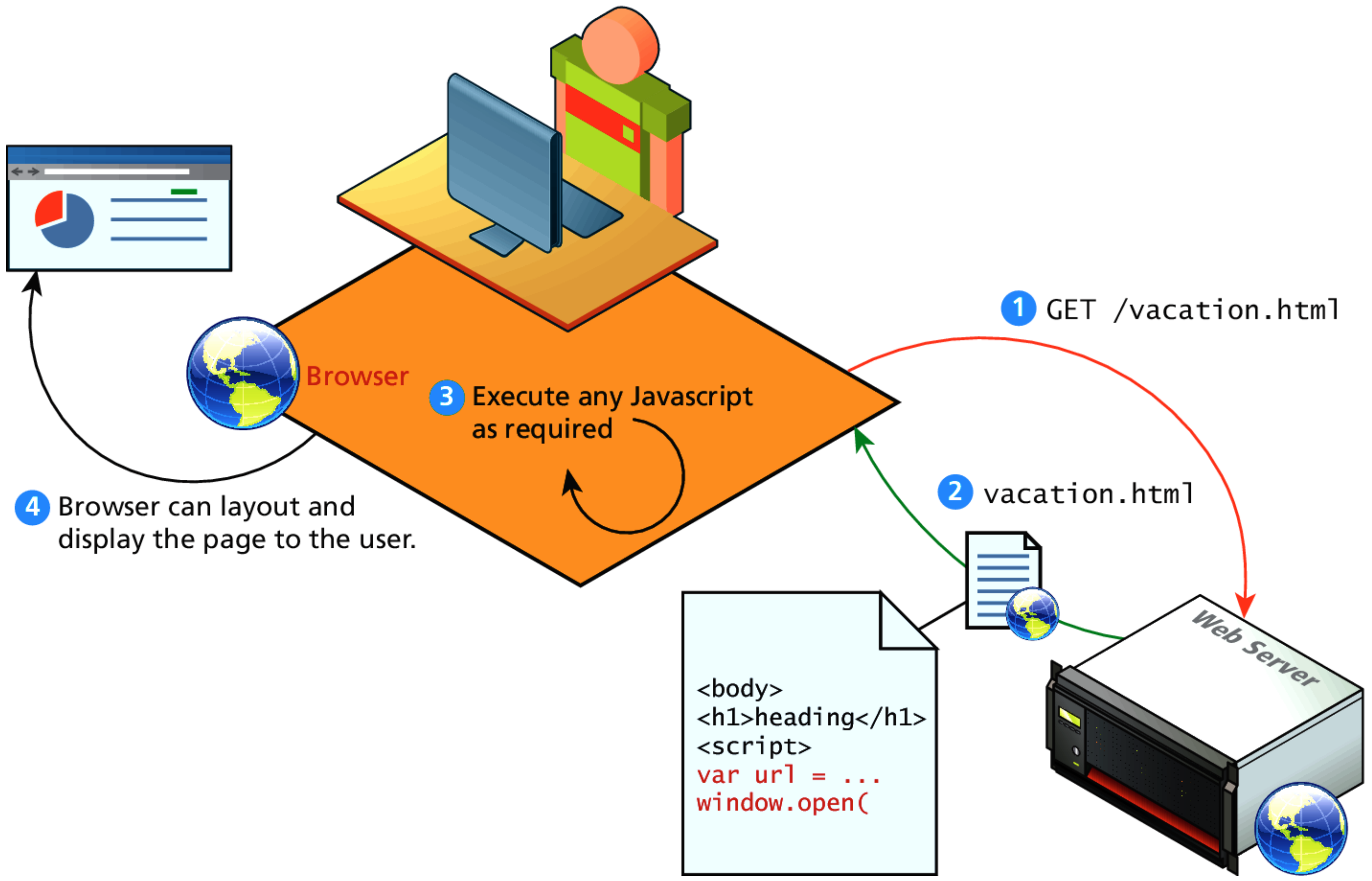
```
window.onloa
  $('start')
  $('end').o
  var bounds
  for (var i
    boundary
```

```
<?php
include('header.p
if (isset($_GET['
  page = $_GET['p
} else {
```

```
<!DOCTYPE html PU
<html xmlns="http
  <head>
    <title>Hello
```

Client-Side Scripting

Let the client compute



Browser
1 GET /vacation.html
2 vacation.html
3 Execute any Javascript as required
4 Browser can layout and display the page to the user.
Web Server

```
<body>
<h1>heading</h1>
<script>
var url = ...
window.open(
```

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.

- JavaScript can interact with the downloaded HTML in a way that the server cannot, creating a user experience more like desktop software than simple HTML ever could.

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
- The idiosyncrasies between various browsers and operating systems make it difficult to test for all potential client configurations. What works in one browser, may generate an error in another.
- JavaScript-heavy web applications can be complicated to debug and maintain.

Why use client-side programming?

PHP already allows us to create dynamic web pages. Why also use client-side scripting?

- client-side scripting (JavaScript) benefits:
  - **usability**: can modify a page without having to post back to the server (faster UI)
  - **efficiency**: can make small, quick changes to page without waiting for server
  - **event-driven**: can respond to user actions like clicks and key presses

Why use client-side programming?

- Server-side programming (PHP) benefits:
  - **security**: has access to server's private data; client can't see source code
  - **compatibility**: not subject to browser compatibility issues
  - **power**: can write files, open connections to servers, connect to databases, ...

What is Javascript?

- A lightweight programming language ("scripting language")
  - used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - **react to events** (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)

What is Javascript?

A web standard (but not supported identically by all browsers)

NOT related to Java other than by name and some syntactic similarities

Javascript vs Java

- interpreted, not compiled
- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- key construct is the function rather than the class
  - "first-class" functions are used in many situations
- contained within a web page and integrates with its HTML/CSS content

Javascript vs Java

Java

+

=

Javascript

JavaScript vs. PHP

- similarities:
  - both are interpreted, not compiled
  - both are relaxed about syntax, rules, and types
  - both are case-sensitive
  - both have built-in regular expressions for powerful text processing

JavaScript vs. PHP

- differences:
  - JS is more object-oriented: noun.verb(), less procedural: verb(noun)
  - JS focuses on user interfaces and interacting with a document; PHP is geared toward HTML output and file/form processing
  - JS code runs on the client's browser; PHP code runs on the web server

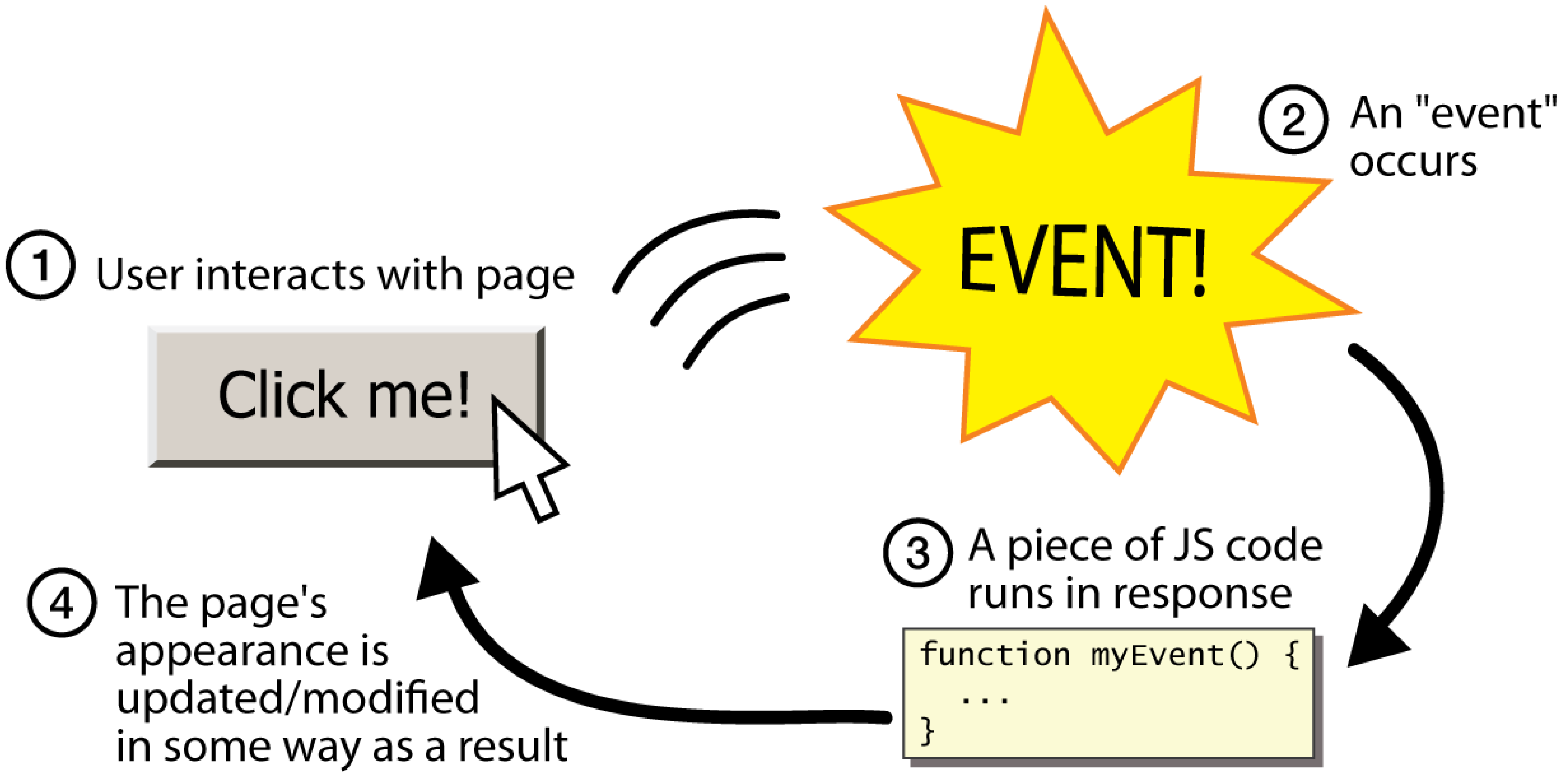
JS <3

Linking to a JavaScript file: `script`

```
<script src="filename" type="text/javascript"></script>
```
HTML

- script tag should be placed in HTML page's head
- script code is stored in a separate .js file
- JS code can be placed directly in the HTML file's body or head (like CSS)
  - but this is bad style (should separate content, presentation, and behavior

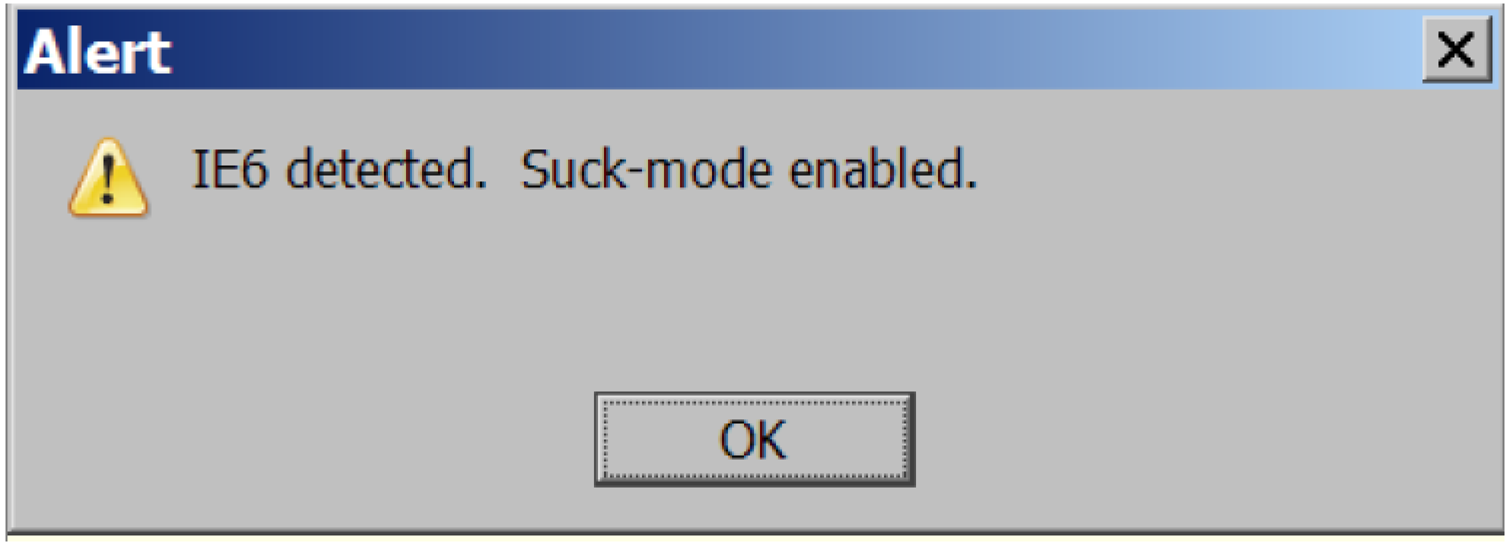
Event-driven programming


1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of JS code runs in response
4 The page's appearance is updated/modified in some way as a result

```
function myEvent() {
  ...
}
```

A JavaScript statement: `alert`

```
alert("IE6 detected. Suck-mode enabled.");
```
JS


Alert

IE6 detected. Suck-mode enabled.

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

Event-driven programming

you are used to programs start with a main method (or implicit main like in PHP)

JavaScript programs instead wait for user actions called *events* and respond to them

event-driven programming: writing programs driven by user events

Let's write a page with a clickable button that pops up a "Hello, World" window...

Buttons

```
<button>Click me!</button>                                   HTML
```

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse 1. click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control

Javascript basic code

```
<html>
<body>
<script language="javascript""type="text/javascript">
<!--
Document.write("hello world")
//-->
</script>
</body>
</html>
```

- <script …….>
- Javascript code
- </script>

Where does JavaScript go?

- JavaScript can be linked to an HTML page in a number of ways.
  - Inline
  - Embedded
  - External

Inner Html

```
<!DOCTYPE html>
<body>
<h1> html page</h1>
<p>fvfvffbgf</p>
<p id =“prog1”</p>
<script>
document.getElementsByID(“prog1”).innerHTML= 6+8;
</script>
</body>
<html>
```

Document write

```
<!DOCTYPE html>
<head>
<script>
document.write(6+8);
</script>
</head>
<body>
<h1> html page</h1>
<p>my paragraph</p>
<script>
document.write(8+8);
</script>
</body>
<html>
```

Document write

```
<!DOCTYPE html>
<body>
<h1> html page</h1>
<p>my paragraph</p>
<button type=“button” onclick=“document.write(6+8)”>Addition</button>
</body>
<html>
```

Window.alert()

```
<!DOCTYPE html>
<body>
<h1> html page</h1>
<p>my paragraph</p>
<script>
alert(6+8);
</script></body>
<html>
```

Javascript Script

```
<!DOCTYPE html>
<body>
<h1> html page</h1>
<p>my paragraph</p>
<button  onlick=“window.print()”>Print this page</button>
</body>
<html>
```

Introduction to Scripting

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

```
<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

```
<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.

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

LISTING 6.3 External JavaScript example

Greetings.js

- Function greeting(){
- Alert(“happy birthday”);
- }

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**.

JavaScript’s Reputation

Precedes it?

JavaScript’s reputation for being quirky not only stems from its strange way of implementing object-oriented principles but also from some odd syntactic *gotchas:*

- Everything is type sensitive, including function, class, and variable names.
- The scope of variables in blocks is not supported. This means variables declared inside a loop may be accessible outside of the loop, counter to what one would expect.
- There is a === operator, which tests not only for equality but type equivalence.
- **Null** and **undefined** are two distinctly different states for a variable.
- Semicolons are not required, but are permitted (and encouraged).
- There is no integer type, only number, which means floating-point rounding errors are prevalent even with values intended to be integers.

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
Assignment

```
var x;      ←  a variable x is defined

var y = 0;  ←  y is defined and initialized to 0

y = 4;      ←  y is assigned the value of 4


/* x conditional assignment */
x = (y==4) ? "y is 4" : "y is not 4";
    ──────   ────────   ────────────
  Condition   Value        Value
              if true      if false
```

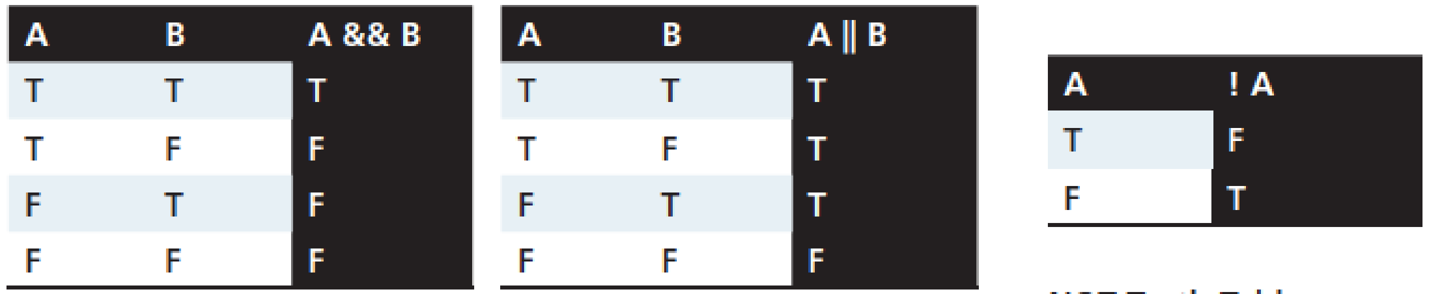
Comparison Operators

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

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

Number type

```
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);
                                                    JS
```

- integers and real numbers are the same type (no int vs. double)
- same operators: + - * / % ++ -- = += -= *= /= %=
- similar precedence to Java
- many operators auto-convert types: "2" * 3 is 6

Comments (same as Java)

```
// single-line comment
/* multi-line comment */
                                                    JS
```

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: <!-- comment -->
  - CSS/JS/PHP: /* comment */
  - Java/JS/PHP: // comment
  - PHP: # comment

Math object

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

- **methods:** `abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan`
- **properties:** `E, PI`

Special values: null and undefined

```
var ned = null;
var benson = 9;
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```
JS

- `undefined` : has not been declared, does not exist
- `null` : exists, but was specifically assigned an empty or null value
- Why does JavaScript have both of these?

Boolean type

```
var iLike190M = true;
var ieIsGood = "IE6" > 0; // false
if ("web devevelopment is great") { /* true */ }
if (0) { /* false */ }
                                                         JS
```

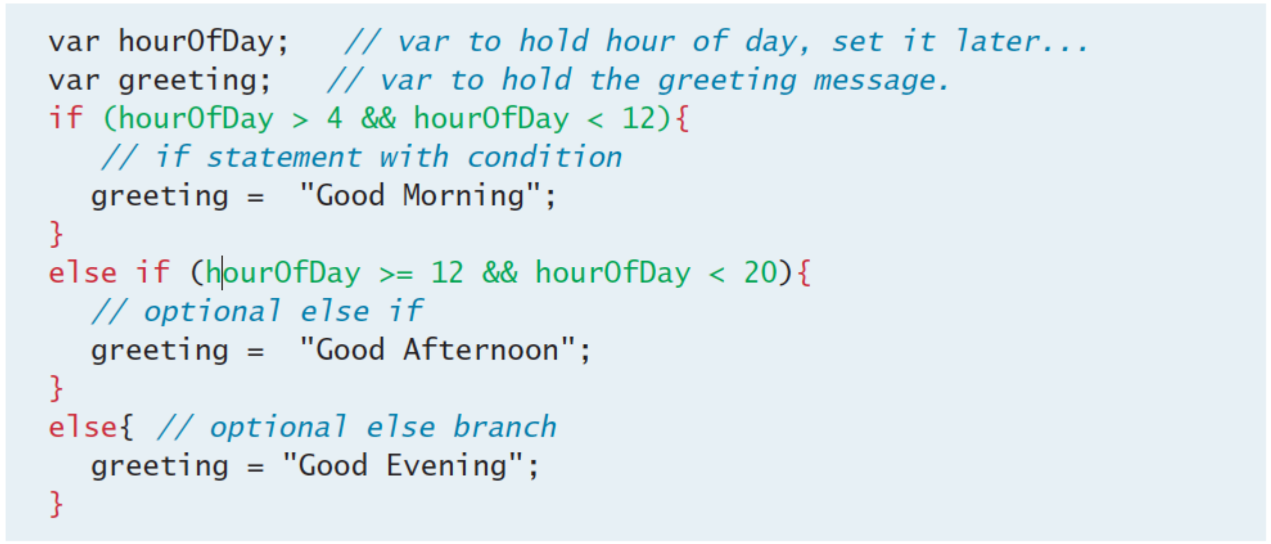
- any value can be used as a Boolean
  - "falsey" values: 0, 0.0, NaN, "", null, and undefined
  - "truthy" values: anything else
- converting a value into a Boolean explicitly:
  - `var boolValue = Boolean(otherValue);`
  - `var boolValue = !!(otherValue);`

Control Statements

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";
}
```

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

if/else statement (same as Java)

```
if (condition) {
        statements;
} else if (condition) {
        statements;
} else {
        statements;
}
                                                      JS
```

- identical structure to Java's if/else statement
- JavaScript allows almost anything as a condition

Loops

Round and round we go

Like conditionals, loops use the ( ) and { } blocks to define the condition and the body of the loop.

You will encounter the **while** and **for** loops

While loops normally initialize a **loop control variable** before the loop, use it in the condition, and modify it within the loop.

```
var i=0;  // initialise the Loop Control Variable

while(i < 10){ //test the loop control variable

        i++;  //increment the loop control variable

}
```

for loop (same as Java)

```
var sum = 0;
for (var i = 0; i < 100; i++) {
       sum = sum + i;
}                                                   JS
```

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
       s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheelllloo"                          JS
```

while loops (same as Java)

```
while (condition) {
        statements;
}                                                                          JS
```

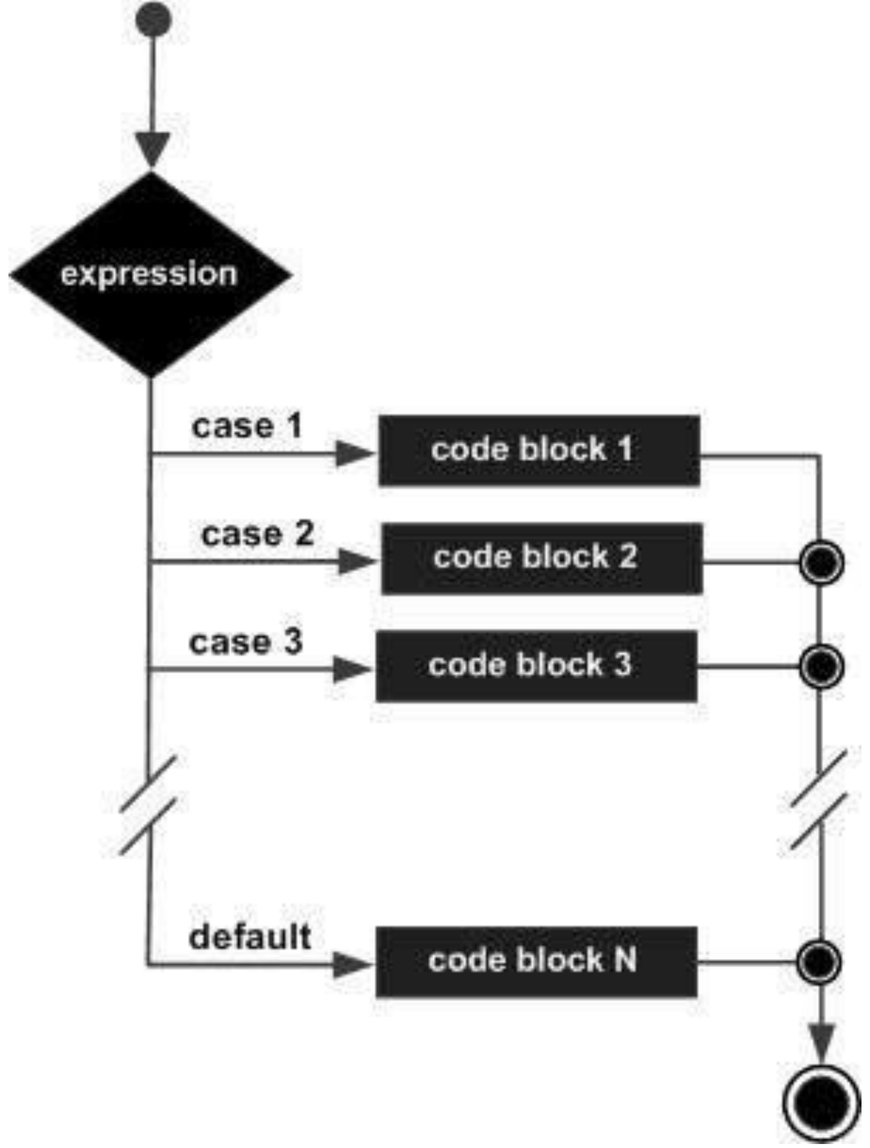
```
do {
    statements;
} while (condition);
                                                                           JS
```

- break and continue keywords also behave as in Java

Switch

```
Switch(expression){
Case condition 1: statement(s);
Break;
Case condition 2: statement(s);
Break;
Case condition 3: statement(s);
Break;
....
Case condition n: statement(s);
break;

default: statement(s)
```



Switch

- <html>
- <body>
- <script>
- Var grade=“’A’’;
- Switch(grade){
- Case ‘’A’’: document.write(“Excelent”);
- Case ‘’B’’: document.write(“Very good”);
- Deafult document.write(“grade not provided”)
- }
- </script>
- </head>
- <html>

JavaScript: Functions

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.

JavaScript functions

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

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

- the above could be the contents of example.js linked to our HTML page
- statements placed into functions can be evaluated in response to user events

Functions

Syntax

```
<script type = "text/javascript">
  <!--
    function function_name(parameter-list) {
      statements
    }
  //-->
</script>
```

```
<script type = "text/javascript">
  <!--
    function Hello() {
      alert("Hello there");
    }
  //-->
</script>
```

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);

Calling a Functions

Example

```
<html>
  <head>
    <script type = "text/javascript">
      function sayHello() {
        document.write ("Hello there!");
      }
    </script>
      </head>
    <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type = "button" onclick = "sayHello()" value = "Say Hello">
    </form>
    <p>Use different text in write method and then try...</p>
  </body>
</html>
```

Function Parameters

Example

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello(name, age) {
            document.write (name + " is " + age + " years old.");
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "sayHello('Zara', 7)" value = "Say Hello">
      </form>
      <p>Use different parameters inside the function and then try...</p>
   </body>
</html>
```

Functions: Return Statement

Example

```
<html>
   <head>
      <script type = "text/javascript">
         function concatenate(first, last) {
            var full;
            full = first + last;
            return full;
         }
         function secondFunction() {
            var result;
            result = concatenate('Zara', 'Ali');
            document.write (result );
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "secondFunction()" value = "Call Function">
      </form>
      <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

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.

Event handlers

```
<element attributes onclick="function();">...
                                                          HTML
```

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

- JavaScript functions can be set as event handlers
  - when you interact with the element, the function will execute
- onclick is just one of many event HTML attributes we'll use
- but popping up an alert window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...

JavaScript Objects

Not full-fledged O.O.

Objects can have **constructors**, **properties**, and **methods** associated with them.

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

```
<script>

Emp={id: 1, name: “A”, salary: 4000}

Var emp=new object();

emp.id= 1;

emp.name= “A”;

emp. Salary= 4000;

document.write(emp.id+””+emp.name+emp.salary);

</script>
```

Constructors

Normally to 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");
```

Constructors

- <html>
- <head>
- <script type=“text/javascript”>
- var book =new object();
- Book.subject=“dbms”;
- Book.author=“korth”
- </script></head>
- <body>
- <script>
- Document.write(“book name”+book.subject+”author is”+book.author)
- </script></body></html>

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.*

Methods

- <html>
- <head>
- <script type=“text/javascript”>
- Function addPrice(amount){
- This.price=amount;}
- Function book(subject,author);{
- this.subject= subject;
- this.author=author;
- This.addPrice=addPrice;} // Assigning that method as property
- </script></head>
- <body>
- <script>
- Var mybook=new book(“dbms”, “korth”);
- Mybook.addPrice(100);
- Document.write(“book name”+mybook.subject+”author is”+mybook.author+”price”+mybook.addprice)
- </script></body></html>

- With(object){
- Properties
- }

- Function addPrice(amount){
- With(this){
- Price=amount;}}

Objects Included in JavaScript

A number of useful objects are included with JavaScript including:

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

Avoid

- X= new Sting();
- Y=new Number();
- Z=new Boolean();

JavaScript: Arrays

Arrays

Arrays are one of the most used data structures. In practice, this class is defined to behave more like a linked list in that it can be resized dynamically, but the implementation is browser specific, meaning the efficiency of insert and delete operations is unknown.

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

```
var greetings = new Array();

Var fruits=new Array(“apple”,”orange”,”mango”,”pear”);

Var fruits=[“apple”,”orange”,”mango”,”pear”];

Var fruits[];

Fruits[0]=“apple”;

Fruits[1]=“banana”

Document.getElementByID(“demo”).innerHTML=fruits;
```

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");
```

or, using the square bracket notation:

```
var greetings = ["Good Morning", "Good Afternoon"];
```

- **Access last element**
- Fruits[fruits.length-1]
- let fruit_length=Fruits.length;
- **Traverse an array:**
- Text=“<ul>”
- For(let i=0;i<fruit_length; i++)
- { text=text+”<li>”+fruit[i]+”</li>”
- }
- Text+=“</ul>”
- -------------------------

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:

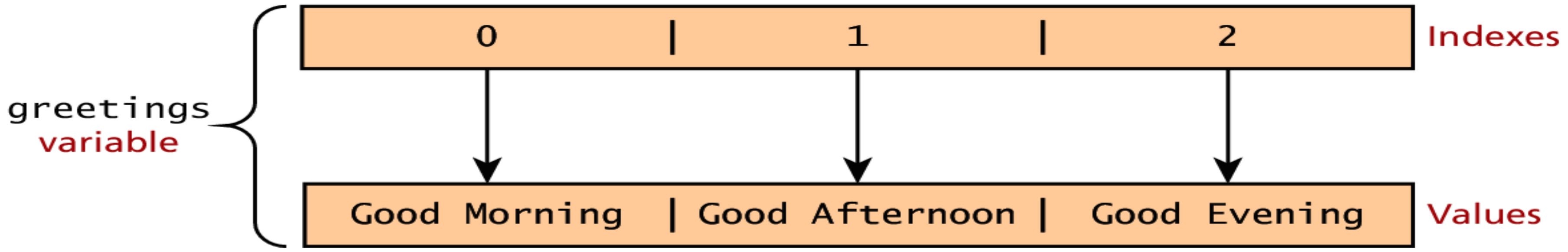
```
for (var i = 0; i < greetings.length; i++){

        alert(greetings[i]);

}
```

Arrays

Index and Value



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

```
var name = []; // empty array
var name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element
                                                        JS
```

```
var ducks = ["Huey", "Dewey", "Louie"];
var stooges = []; // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
                                                        JS
```

Array methods

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
                                                          JS
```

- array serves as many data structures: list, queue, stack, ...
- methods: `concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift`
  - push and pop add / remove from back
  - unshift and shift add / remove from front
  - shift and pop return the element that is removed

`String` type

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
var s2 = 'Melvin Merchant';
                                                          JS
```

- methods: `charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase`
- charAt returns a one-letter String (there is no char type)
- length property (not a method as in Java)
- Strings can be specified with "" or ''
- concatenation with + :
- 1 + 1 is 2, but "1" + 1 is "11"

More about `String`

- escape sequences behave as in Java: \' \" \& \n \t \\
- converting between numbers and Strings:

```
var count = 10;
var s1 = "" + count; // "10"
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah
ah ah!"
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("booyah"); // NaN                     JS
```

- accessing the letters of a String:

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);                  JS
```

Splitting strings: split and join

```
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
                                                        JS
```

- split breaks apart a string into an array using a delimiter
  - can also be used with regular expressions (seen later)
- join merges an array into a single string, placing a delimiter between them

Document object Model (DOM)

The DOM
Document Object Model

JavaScript is almost always used to interact with the HTML document in which it is contained.

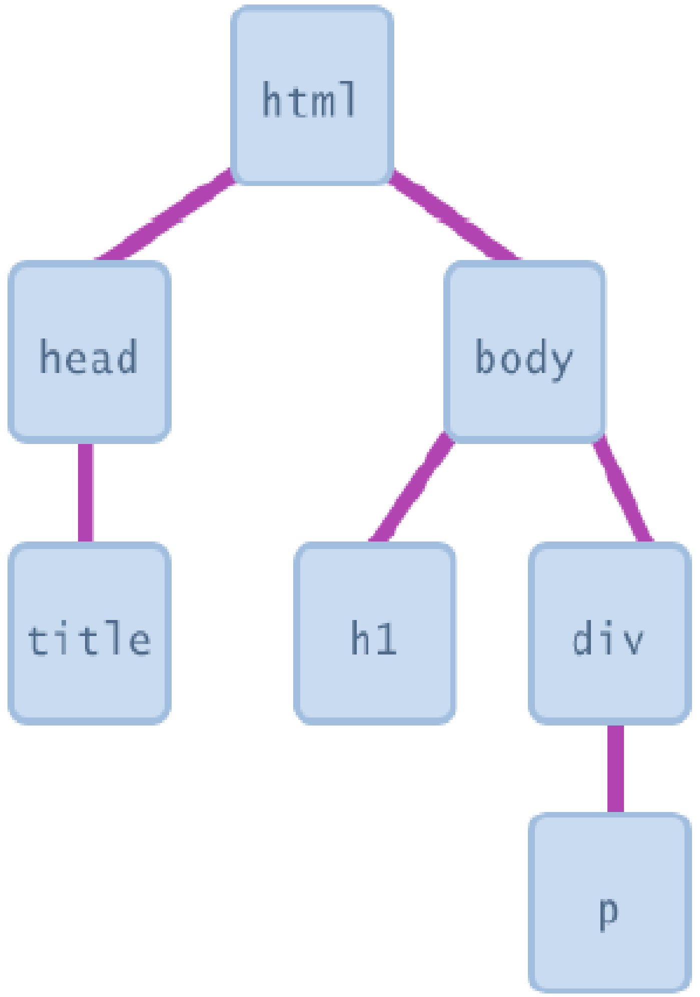
This is accomplished through a programming interface (API) called the **Document Object Model.**

According to the W3C, the DOM is a:

Platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

Document Object Model (DOM)

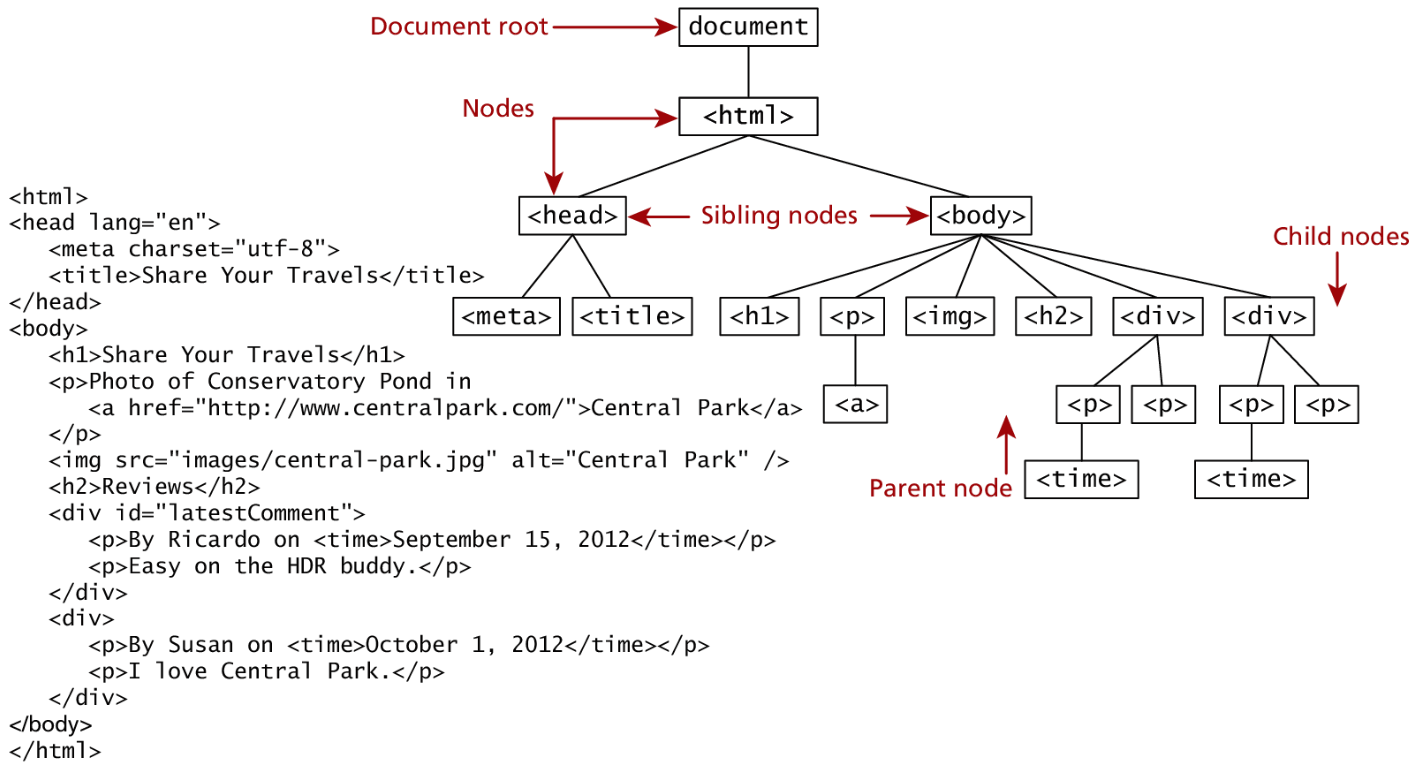
- most JS code manipulates elements on an HTML page
- we can examine elements' state
  - e.g. see whether a box is checked
- we can change state
  - e.g. insert some new text into a div
- we can change styles
  - e.g. make a paragraph red



The DOM

Seems familiar, because it is!

The Document Object Model(DOM) is the objects that make up a web page. This tree structure is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.



```
<html>
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>September 15, 2012</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```

DOM Nodes

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:

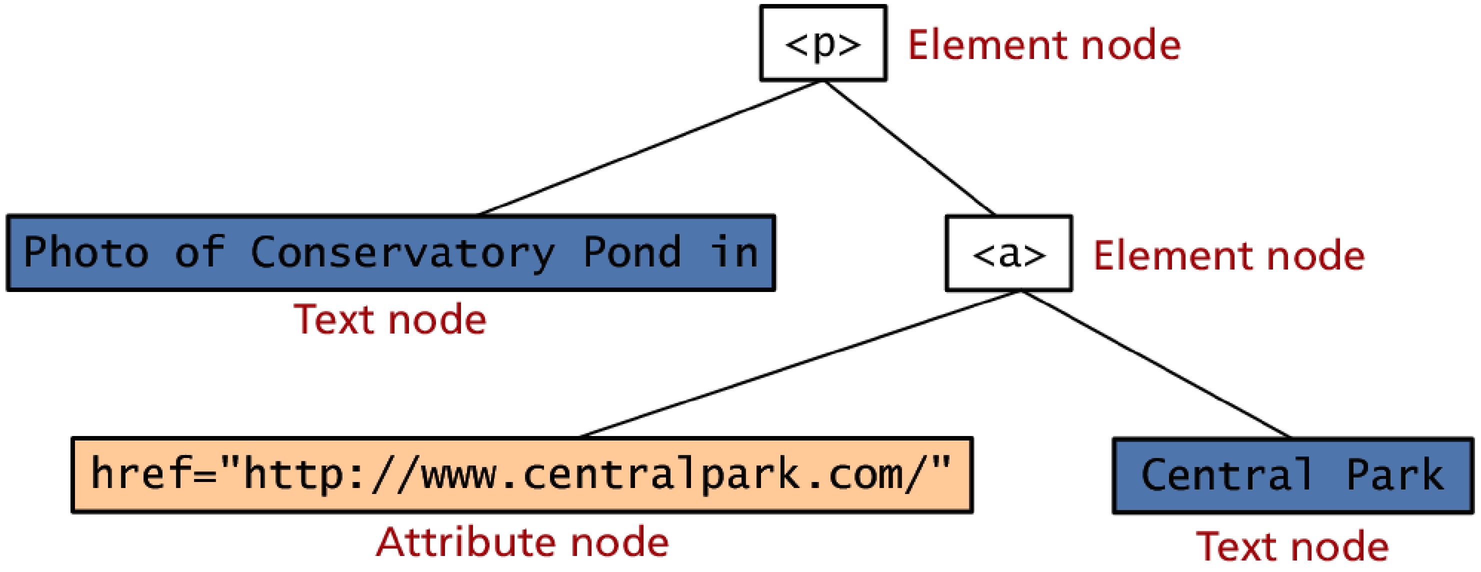
- element nodes,
- text nodes, and
- attribute nodes

All nodes in the DOM share a common set of properties and methods.

DOM Nodes

Element, text and attribute nodes

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



DOM Nodes

Essential Node Object properties

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

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

var b = document.inputEncoding;
```

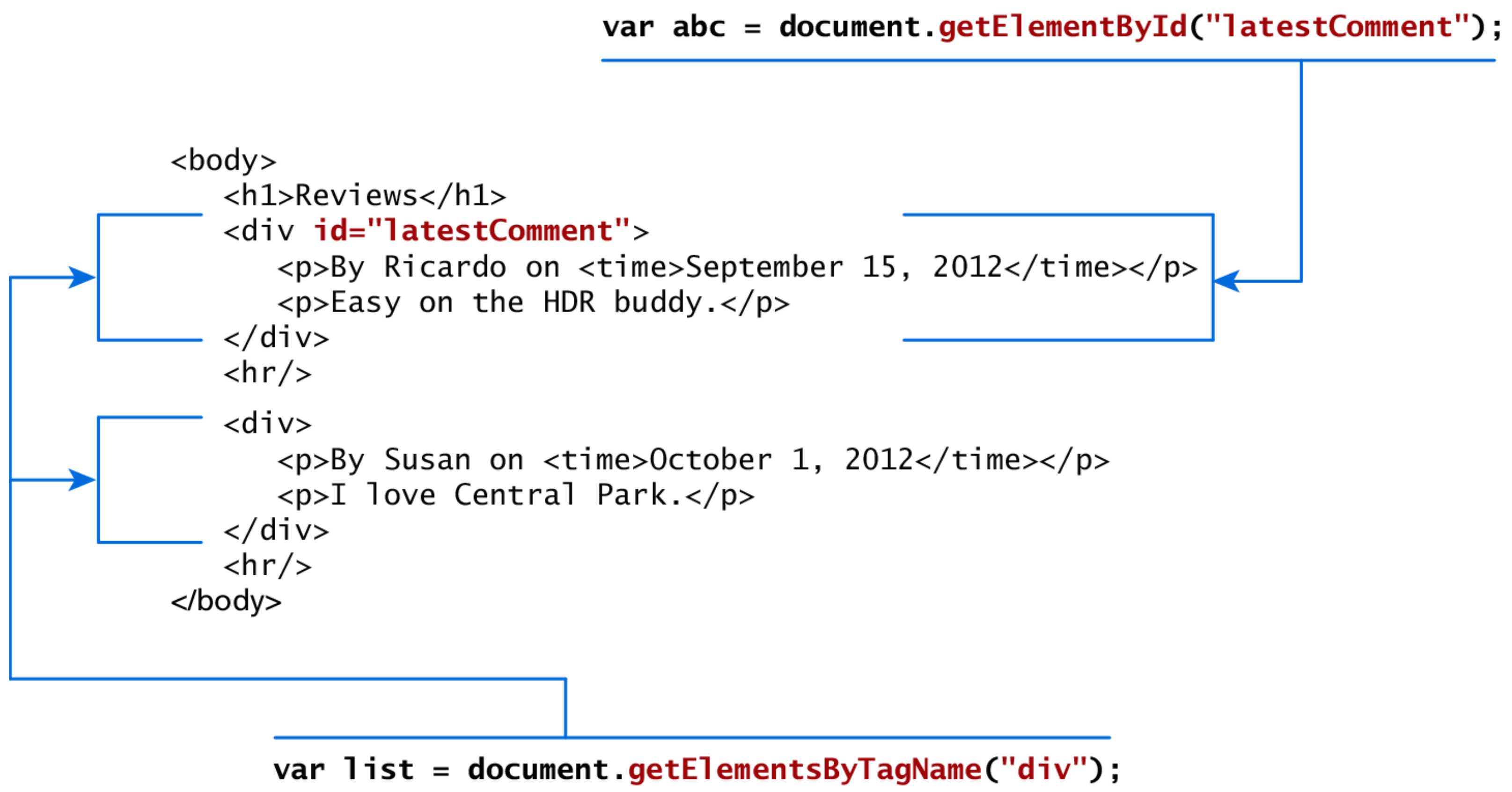
Document Object

Document Object Methods

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

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. The modern JavaScript programmer will want to write to the HTML page, but in a particular location, not always at the bottom

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

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.

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.backgroundColour = "#FFFF00";

commentTag.style.borderWidth="3px";
```

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.

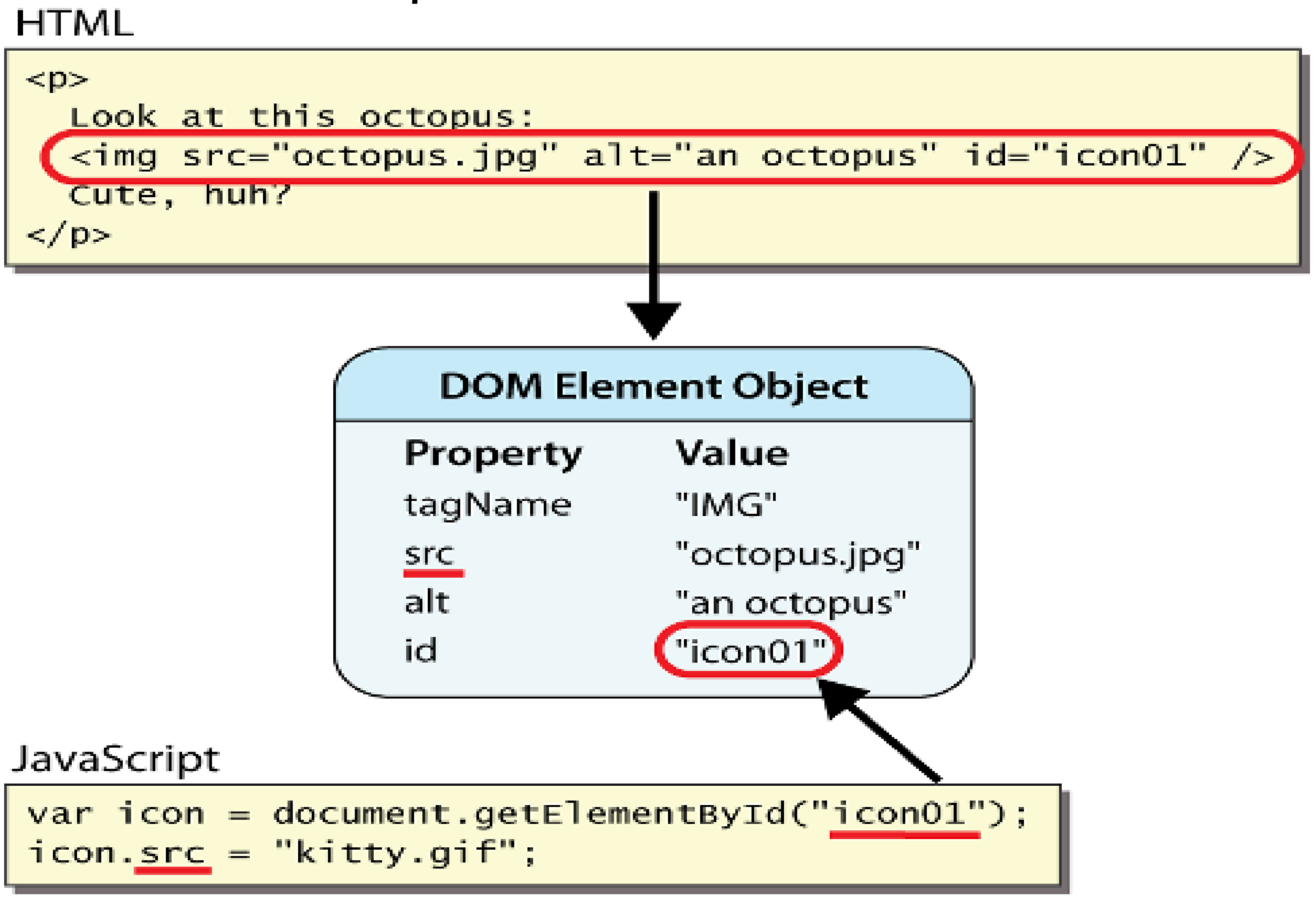
```
label.classList.addClass("someClassName");
```

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

DOM element objects


HTML

```
<p>
  Look at this octopus:
  <img src="octopus.jpg" alt="an octopus" id="icon01" />
  Cute, huh?
</p>
```

Property	Value
tagName	"IMG"
src	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

Accessing elements: `document.getElementById`

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

```
<button onclick="changeText();">Click me!</button>
<span id="output">replace me</span>
<input id="textbox" type="text" />                           HTML
```

```
function changeText() {
        var span = document.getElementById("output");
        var textBox = document.getElementById("textbox");

         textbox.style.color = "red";

}                                                               JS
```

Accessing elements: `document.getElementById`

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

Changing element style: `element.style`

Attribute	**Property or style object**
color	color
padding	padding
background-color	backgroundColor
border-top-width	borderTopWidth
Font size	fontSize
Font famiy	fontFamily

Preetify

```
function changeText() {
        //grab or initialize text here

        // font styles added by JS:
        text.style.fontSize = "13pt";
        text.style.fontFamily = "Comic Sans MS";
        text.style.color = "red"; // or pink?
}
```

JS

DOM with HTML Forms

-
  ```
  function validateForm() {
    let x = document.forms["myForm"]["fname"].value;
    if (x == "") {
      alert("Name must be filled out");
      return false;
    }
  }
  ```

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

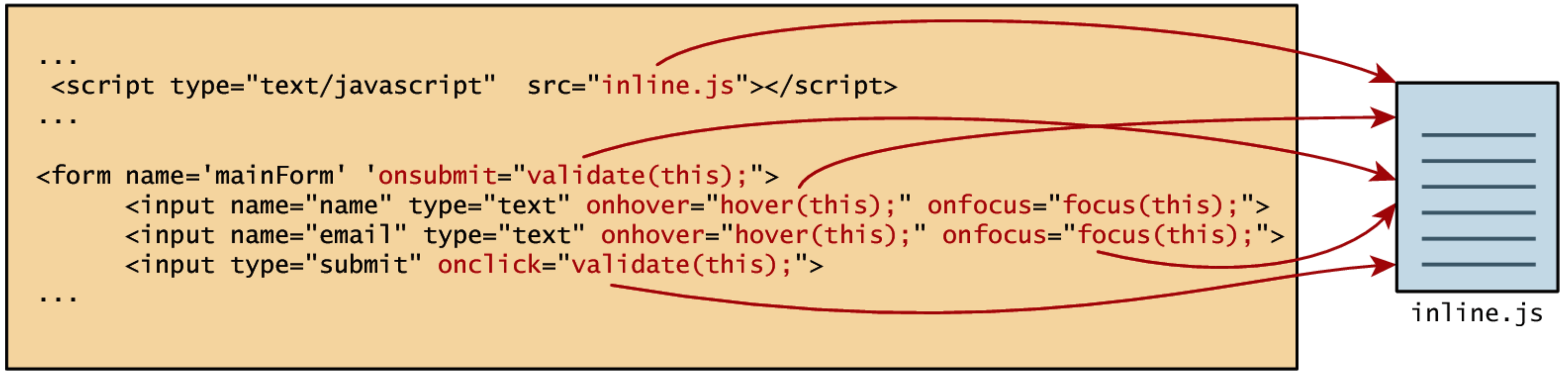
In the original JavaScript world, events could be specified right in the HTML markup with *hooks* to the JavaScript code (and still can).

As more powerful frameworks were developed, and website design and best practices were refined, this original mechanism was supplanted by the **listener** approach.

JavaScript Events

Two approaches

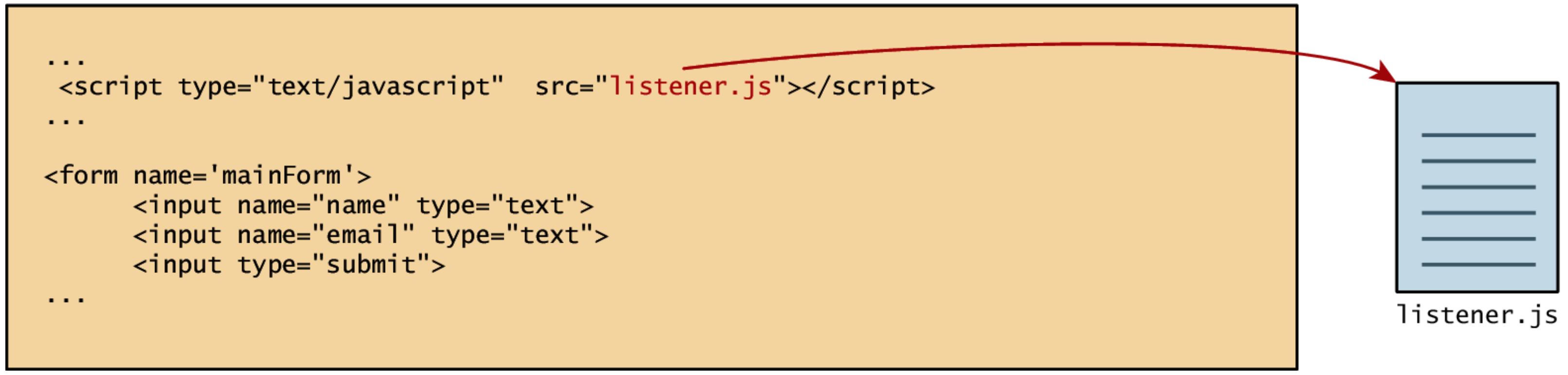
Old, Inline technique



```
...
 <script type="text/javascript"  src="inline.js"></script>
...

<form name='mainForm' 'onsubmit="validate(this);">
      <input name="name" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input name="email" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input type="submit" onclick="validate(this);">
...
```

New, Layered Listener technique



```
...
 <script type="text/javascript"  src="listener.js"></script>
...

<form name='mainForm'>
      <input name="name" type="text">
      <input name="email" type="text">
      <input type="submit">
...
```

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

Using functions

What if we wanted to do something more elaborate when an event is triggered? In such a case, the behavior would have to be encapsulated within a function, as shown in Listing 6.12.

```
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

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

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 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

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

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