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

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 *interpreted 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

Let the client compute

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

```html
<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.
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 was only slightly useful to many users
HTTP request-response loop

**Without JavaScript**

1. GET /form.php
2. Requested page is returned
3. User selects country, then clicks Update button.
4. GET /form.php?country=canada
5. Requested page (with updated form) is returned
6. User continues with form, perhaps triggering other requests...
JavaScript in Modern Times

AJAX

JavaScript became a much more important part of web development in the mid 2000s with AJAX.

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

- As an acronym it means Asynchronous JavaScript And XML.
- The most important feature of AJAX sites is the asynchronous data requests.
Asynchronous data requests
The better AJAX way

1. Request
2. Response
3. After browser receives a response to its HTTP request, it blanks the browser window, and ...
4. ... renders the just-received HTML in the browser window.
5. User clicks update button.
6. Via JavaScript, browser makes asynchronous request for data.
7. Browser returns XML or JSON or some other type of data
8. Via JavaScript, browser dynamically updates window to change label and populate list with provinces of Canada.

Randy Connolly and Ricardo Hoar
Fundamentals of Web Development
Section 2 of 8

JAVASCRIPT DESIGN PRINCIPLES
Layers

They help organize

When designing software to solve a problem, it is often helpful to abstract the solution a little bit to help build a cognitive model in your mind that you can then implement.

Perhaps the most common way of articulating such a cognitive model is via the term layer.

In object-oriented programming, a software layer is a way of conceptually grouping programming classes that have similar functionality and dependencies.
Layers
Common Layers

- **Presentation layer.** Classes focused on the user interface.

- **Business layer.** Classes that model real-world entities, such as customers, products, and sales.

- **Data layer.** Classes that handle the interaction with the data sources.
Layers

Just a conceptual idea
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.

---

**LISTING 6.1** Inline JavaScript example

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

Better

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

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

**LISTING 6.2** Embedded JavaScript example
External JavaScript

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

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

```html
<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

Assignment

```javascript
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";
```

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value if true</th>
<th>Value if false</th>
</tr>
</thead>
</table>
## Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Matches (x=9)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equals</td>
<td>(x==9) is true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x==&quot;9&quot;) is true</td>
</tr>
<tr>
<td><code>===</code></td>
<td>exactly equals, including type</td>
<td>(x===&quot;9&quot;) is false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x===9) is true</td>
</tr>
<tr>
<td><code>&lt;, &gt;</code></td>
<td>less than, greater than</td>
<td>(x&lt;5) is false</td>
</tr>
<tr>
<td><code>&lt;=, &gt;=</code></td>
<td>less than or equal, greater than or equal</td>
<td>(x&lt;=9) is true</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal</td>
<td>(4!=x) is true</td>
</tr>
<tr>
<td><code>!==</code></td>
<td>not equal in either value or type</td>
<td>(x!=&quot;9&quot;) is true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x!==9) is false</td>
</tr>
</tbody>
</table>
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
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
}
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.
Functions

Example

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

```javascript
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:

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

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

```javascript
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

```javascript
try {
  nonexistantfunction("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 "smallerThanZeroError";
}
catch(err){
    alert (err + "was thrown");
}
```

**LISTING 6.6** Throwing a user-defined exception
Section 5 of 8

JAVASCRIPT OBJECTS
JavaScript Objects

Objects not Classes

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

It does not have classes per se, and 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.

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.
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");
```
Properties

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
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:

```javascript
var greetings = new Array();
```
Arrays
Initialize with values

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

```javascript
var greetings = new Array("Good Morning", "Good Afternoon");
```

or, using the square bracket notation:

```javascript
var greetings = ["Good Morning", "Good Afternoon"];  
```
THE 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.
The DOM

Seems familiar, because it is!

We already know all about the DOM, but by another name. The tree structure from Chapter 2 (HTML) is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.
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</p>
<a href="http://www.centralpark.com/">Central Park</a>

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

### Essential Node Object properties

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

```javascript
// 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createAttribute()</code></td>
<td>Creates an attribute node</td>
</tr>
<tr>
<td><code>createElement()</code></td>
<td>Creates an element node</td>
</tr>
<tr>
<td><code>createTextNode()</code></td>
<td>Create a text node</td>
</tr>
<tr>
<td><code>getElementById(id)</code></td>
<td>Returns the element node whose id attribute matches the passed id parameter.</td>
</tr>
<tr>
<td><code>getElementsByTagName(name)</code></td>
<td>Returns a nodeList of elements whose tag name matches the passed name parameter.</td>
</tr>
</tbody>
</table>
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>Easy on the HDR buddy.</p>
</div>
<hr/>

<div>
  <p>By Susan on <time>October 1, 2012</time>
  <p>I love Central Park.</p>
</div>
<hr/>
</body>

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

```javascript
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.

```javascript
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()
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

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

```html
...<script type="text/javascript" src="inline.js"></script>
...
<form name='mainForm' onsubmit="validate(this);">
  <input name='name' type='text' onmouseover="hover(this);" onfocus="focus(this);">
  <input name='email' type='text' onmouseover="hover(this);" onfocus="focus(this);">
  <input type='submit' onclick='validate(this);'>
</form>
...
```

New, Layered Listener technique

```html
...<script type="text/javascript" src="listener.js"></script>
...
<form name='mainForm'>
  <input name='name' type='text'>
  <input name='email' type='text'>
  <input type='submit'>
</form>
...
Inline Event Handler Approach

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

```html
<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.

```javascript
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.

```javascript
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

```javascript
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
## Mouse events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onclick</td>
<td>The mouse was clicked on an element</td>
</tr>
<tr>
<td>ondblclick</td>
<td>The mouse was double clicked on an element</td>
</tr>
<tr>
<td>onmousedown</td>
<td>The mouse was pressed down over an element</td>
</tr>
<tr>
<td>onmouseup</td>
<td>The mouse was released over an element</td>
</tr>
<tr>
<td>onmouseover</td>
<td>The mouse was moved (not clicked) over an element</td>
</tr>
<tr>
<td>onmouseout</td>
<td>The mouse was moved off of an element</td>
</tr>
<tr>
<td>onmousemove</td>
<td>The mouse was moved while over an element</td>
</tr>
</tbody>
</table>
## Keyboard events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onkeydown</td>
<td>The user is pressing a key (this happens first)</td>
</tr>
<tr>
<td>onkeypress</td>
<td>The user presses a key (this happens after onkeydown)</td>
</tr>
<tr>
<td>onkeyup</td>
<td>The user releases a key that was down (this happens last)</td>
</tr>
</tbody>
</table>
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.

```javascript
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
## Form Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onblur</td>
<td>A form element has lost focus (that is, control has moved to a different element, perhaps due to a click or Tab key press.)</td>
</tr>
<tr>
<td>onchange</td>
<td>Some <code>&lt;input&gt;</code>, <code>&lt;textarea&gt;</code> or <code>&lt;select&gt;</code> field had their value change. This could mean the user typed something, or selected a new choice.</td>
</tr>
<tr>
<td>onfocus</td>
<td>Complementing the <code>onblur</code> event, this is triggered when an element gets focus (the user clicks in the field or tabs to it)</td>
</tr>
<tr>
<td>onreset</td>
<td>HTML forms have the ability to be reset. This event is triggered when that happens.</td>
</tr>
<tr>
<td>onselect</td>
<td>When the users selects some text. This is often used to try and prevent copy/paste.</td>
</tr>
<tr>
<td>onsubmit</td>
<td>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.</td>
</tr>
</tbody>
</table>
Form Events

Example

```javascript
document.getElementById("loginForm").onsubmit = function(e){
    var pass = document.getElementById("pw").value;
    if(pass==""){
        alert("enter a password");
        e.preventDefault();
    }
}
```

**LISTING 6.16** Catching the onsubmit event and validating a password to not be blank
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

Empty field

```javascript
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** A simple validation script to check for empty fields
Validating Forms

Empty field

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

```javascript
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
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:

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

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