



# Web Programmi ng

CS-406

Lecture # 04

JavaScript

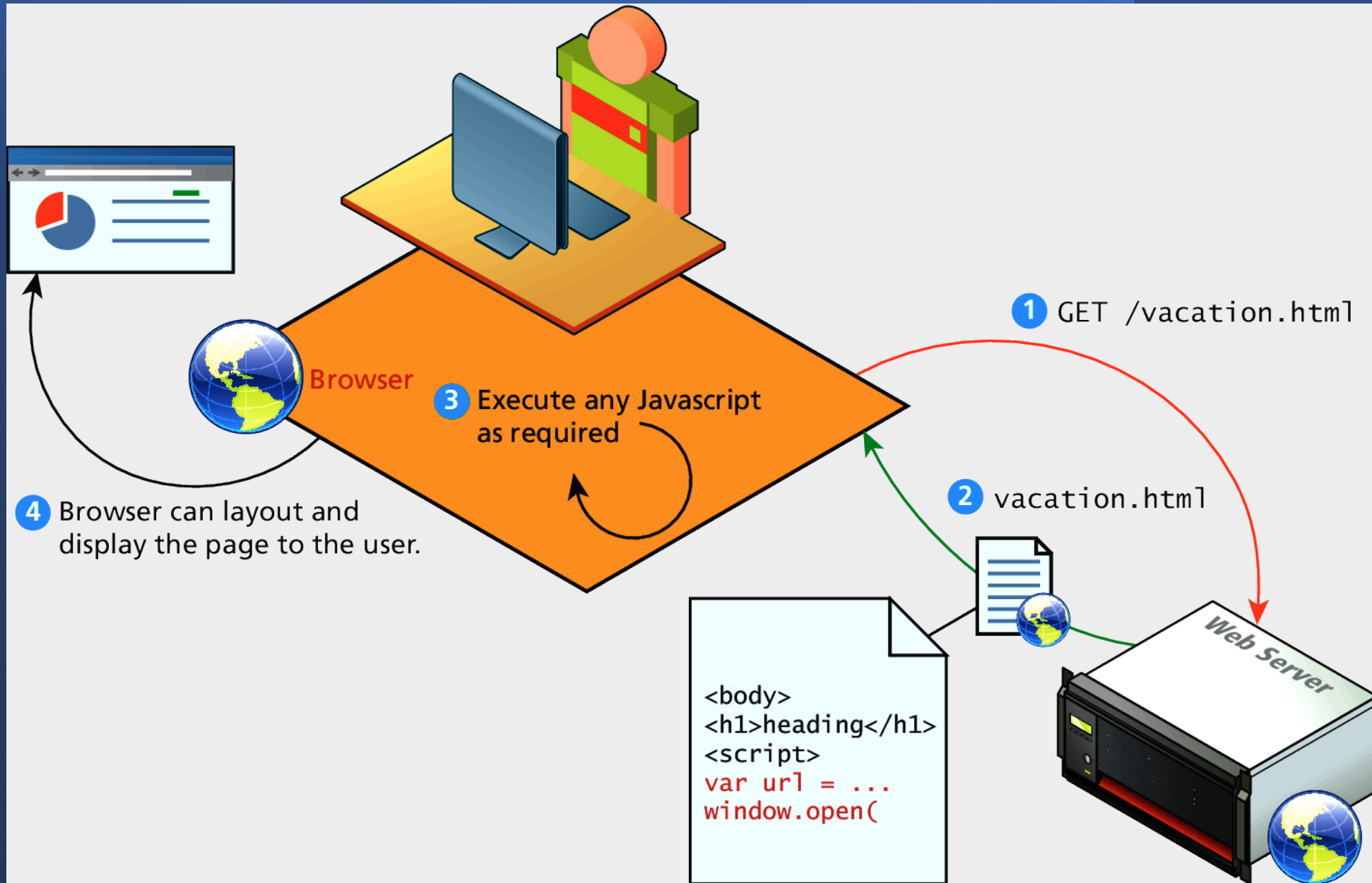
# JavaScript

- High level Interpreted language
- Object oriented approach.
- Enables interactive webpages .
- Used for controlling client-side page behavior.
- Multi-paradigm language: JavaScript supports event-driven, functional, and imperative programming styles.

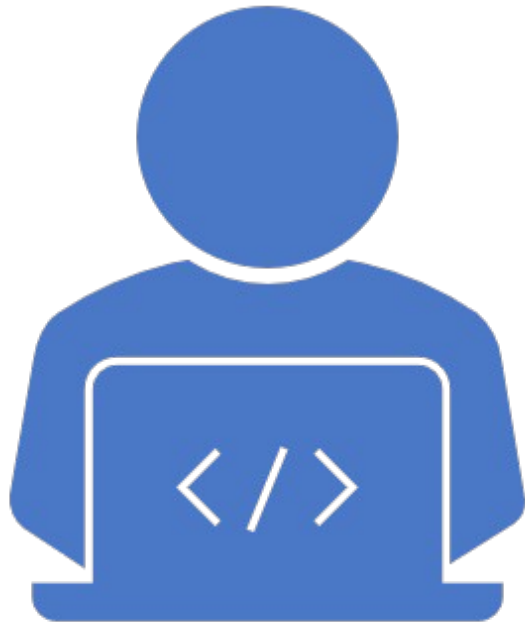
- It has application programming interfaces (APIs) for working with
  - text,
  - dates,
  - regular expressions,
  - standard data structures, and
  - the Document Object Model (DOM).
- However, the language itself does not include any input/output (I/O), such as networking, storage, or graphics facilities, as the host environment (usually a web browser) provides those APIs.

# What is JS?

- 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, python or C#, it is also quite different from those languages



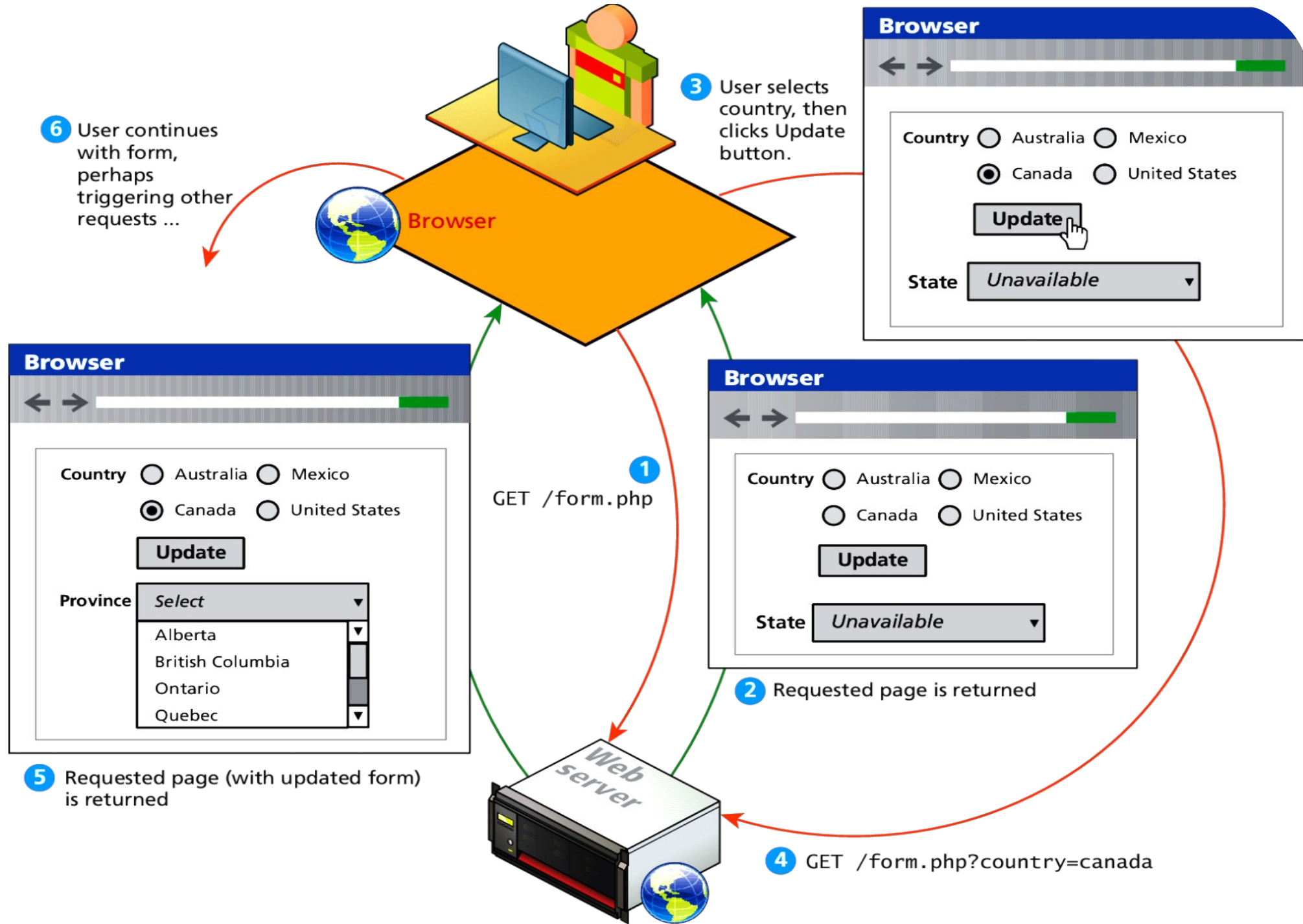
# Client-side scripting



- 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 unconventional behavior 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.
  - Heavy web applications can be complicated to debug and maintain.



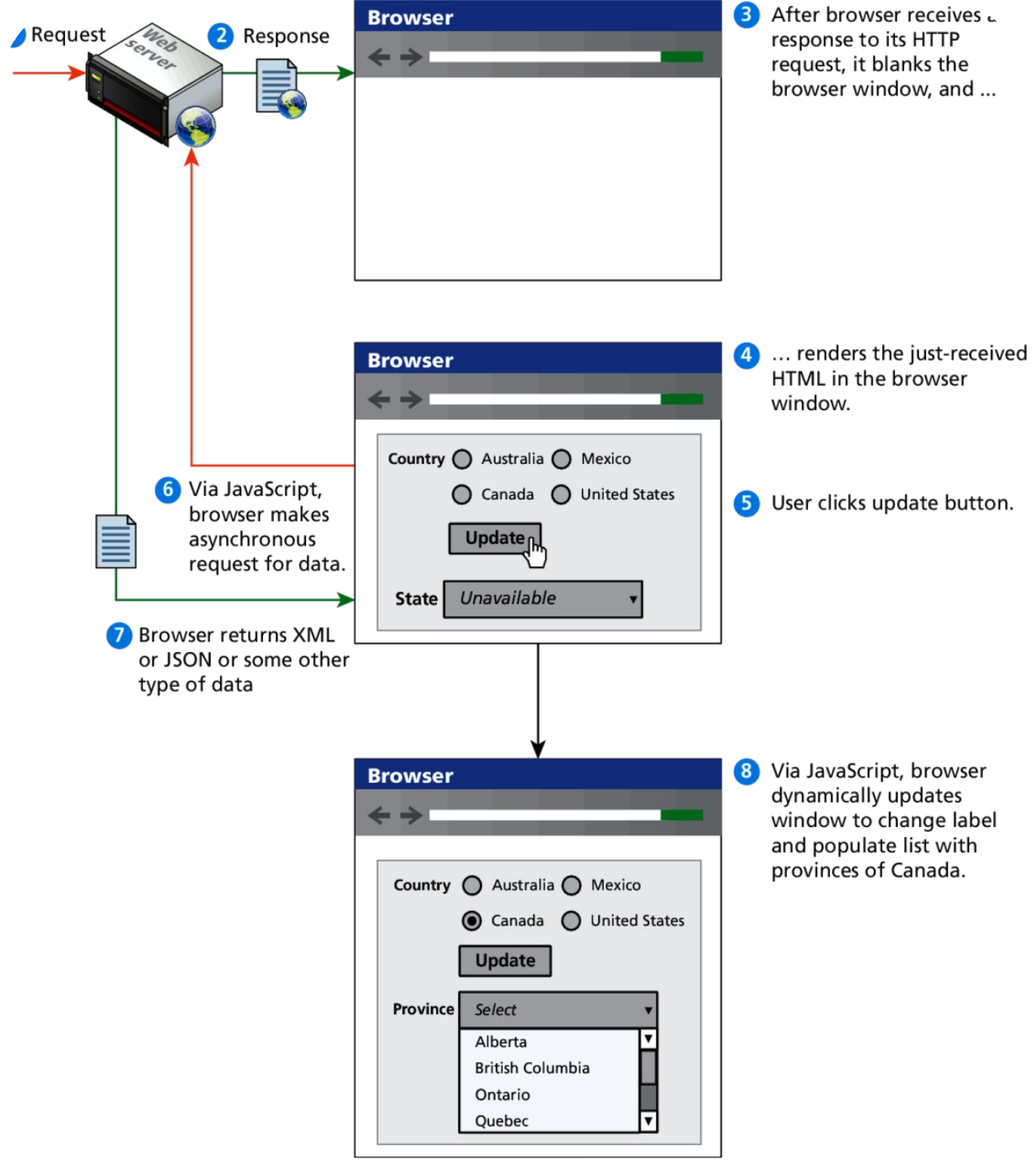
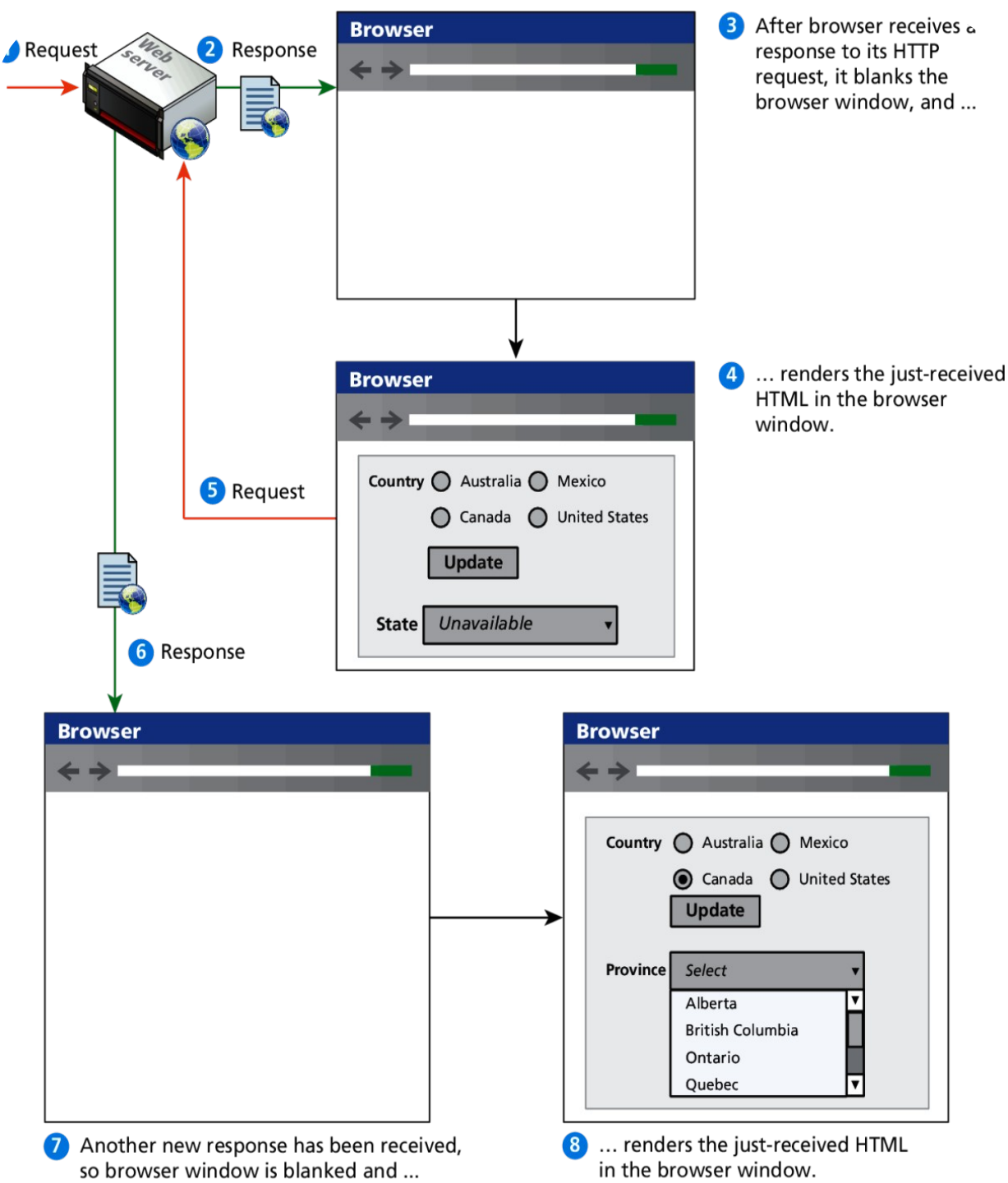
# Forms



# 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 **A**synchronous **J**avaScript **A**nd **X**ML.
- The most important feature of AJAX sites is the asynchronous data requests.





# JAVAScript - HTML link


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

# Inline JavaScript

- Inline JavaScript refers to the practice of including JavaScript code directly within certain HTML attributes
- Maintenance issues.
- `<button onclick="alert('Are you sure?');">`
- `<button onclick="this.innerHTML=Date()">The time is?</button>`

# Embedded JavaScript

- Embedded JavaScript refers to the practice of placing JavaScript code within a `<script>` element
- Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.
- In HTML, JavaScript code must be inserted between **`<script>`** and **`</script>`** tags.

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

# 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.

```
<head>
```

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

```
</head>
```

A dark blue, irregular ink splatter shape is centered on a white background. The splatter has a textured, painterly appearance with various shades of blue and grey. The text "JavaScript syntax" is written in white, sans-serif font, centered within the dark blue area.


# JavaScript syntax



The fundamental syntax for the most common programming constructs including :

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

Advanced topics such as **events** and **classes**.

- 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.
- 

# Variables

- **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 as the familiar type fields like *int*, *char*, and *String* etc aren't required. Instead **var** is used.

`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  
if true


Value  
if false

# Comparison operators

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

# Arithmetic operators

- `=`
- `+=`
- `-=`
- `*=`
- `%=`
- `/=`
- `**=`

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

- ```
var x = 5 + 5;           //10  
var y = "5" + 5;        //55  
var z = "Hello" + 5;     //Hello5
```



# 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

# Conditional Statements

- 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.
- **if (condition) {**  
    *// block of code to be executed if the condition is true*  
**}**

# Loops

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

# While loop

- **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

- A **for loop** combines the common components of a loop: initialization, condition, and post-loop operation into one statement.
- This statement begins with the **for** keyword and has the components placed between ( ) brackets, semicolon (;) separated
- **For(var i=0; i<10; i++)**
- {
- *//Do Something*
- }

# JavaScript- Display

JavaScript can "display" data in different ways:

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

# Functions

- **Functions** are the building block for modular code in JavaScript.
- 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*  
**}**
- function myFunction(p1, p2) {  
    return p1 \* p2; // The function returns the product of p1 and  
    p2  
}

# Alert

- 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.

# JavaScript objects

- JavaScript is not a full-fledged object-oriented programming language.
- It does not have classes , 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.
- 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.

# Object 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 = 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

- 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 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:

```
var greetings = new Array();
```

# Initialization

- 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"];
```

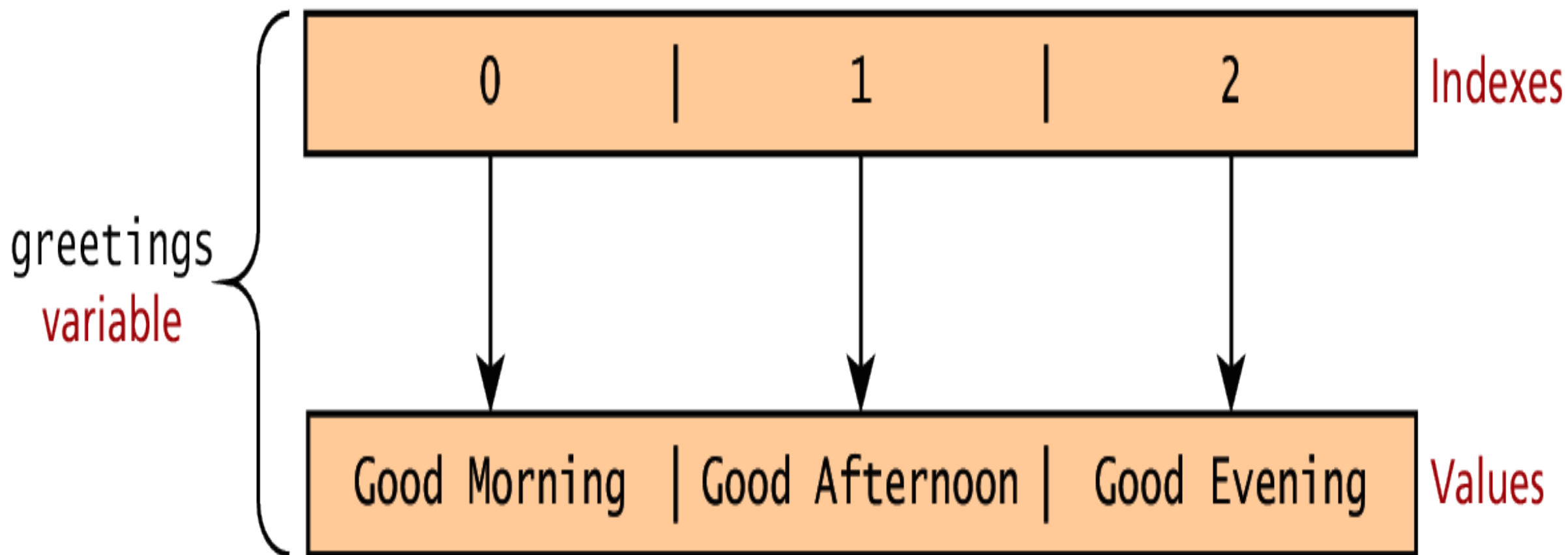
# Accessing values

- 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:

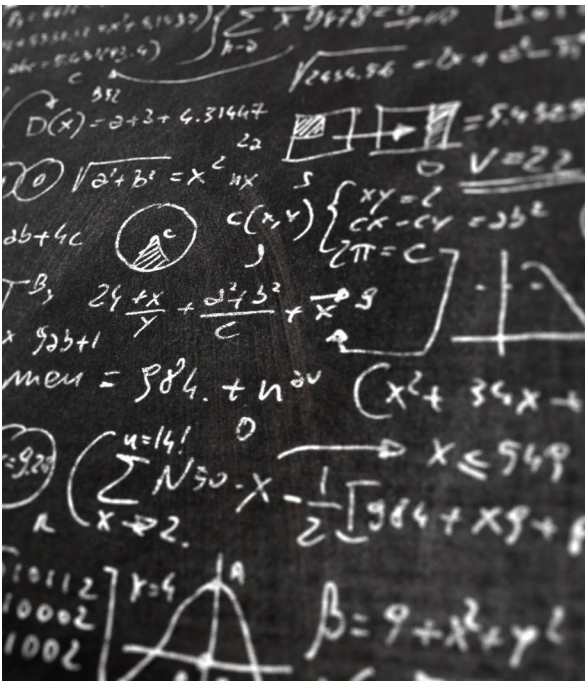
```
for (var i = 0; i < greetings.length; i++){  
    alert(greetings[i]);  
}
```



# Array Methods

- 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()`

# Math



- 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



# 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"*

- `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());`

# 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.

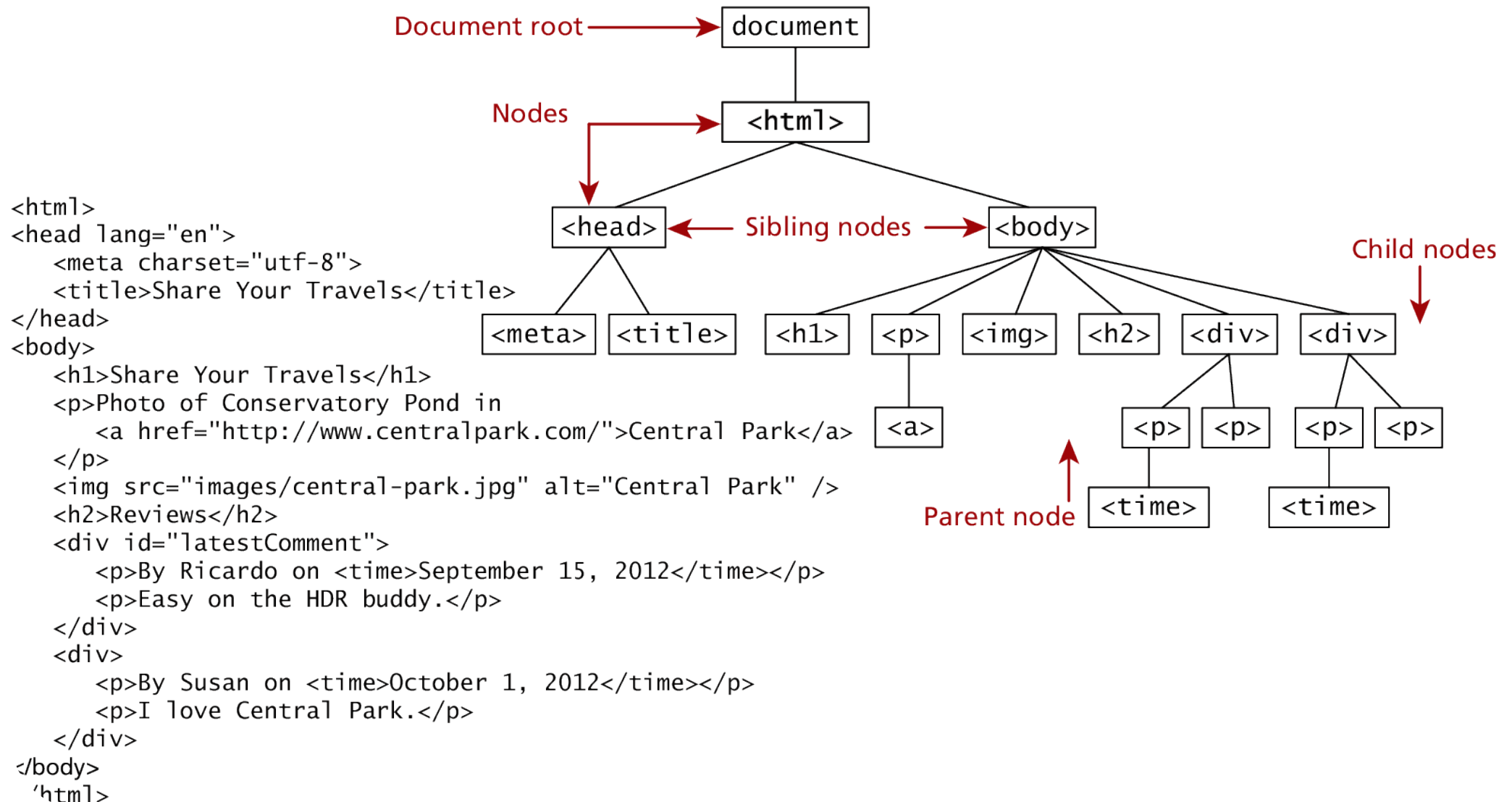
# The Dom (document object model)

- 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.*

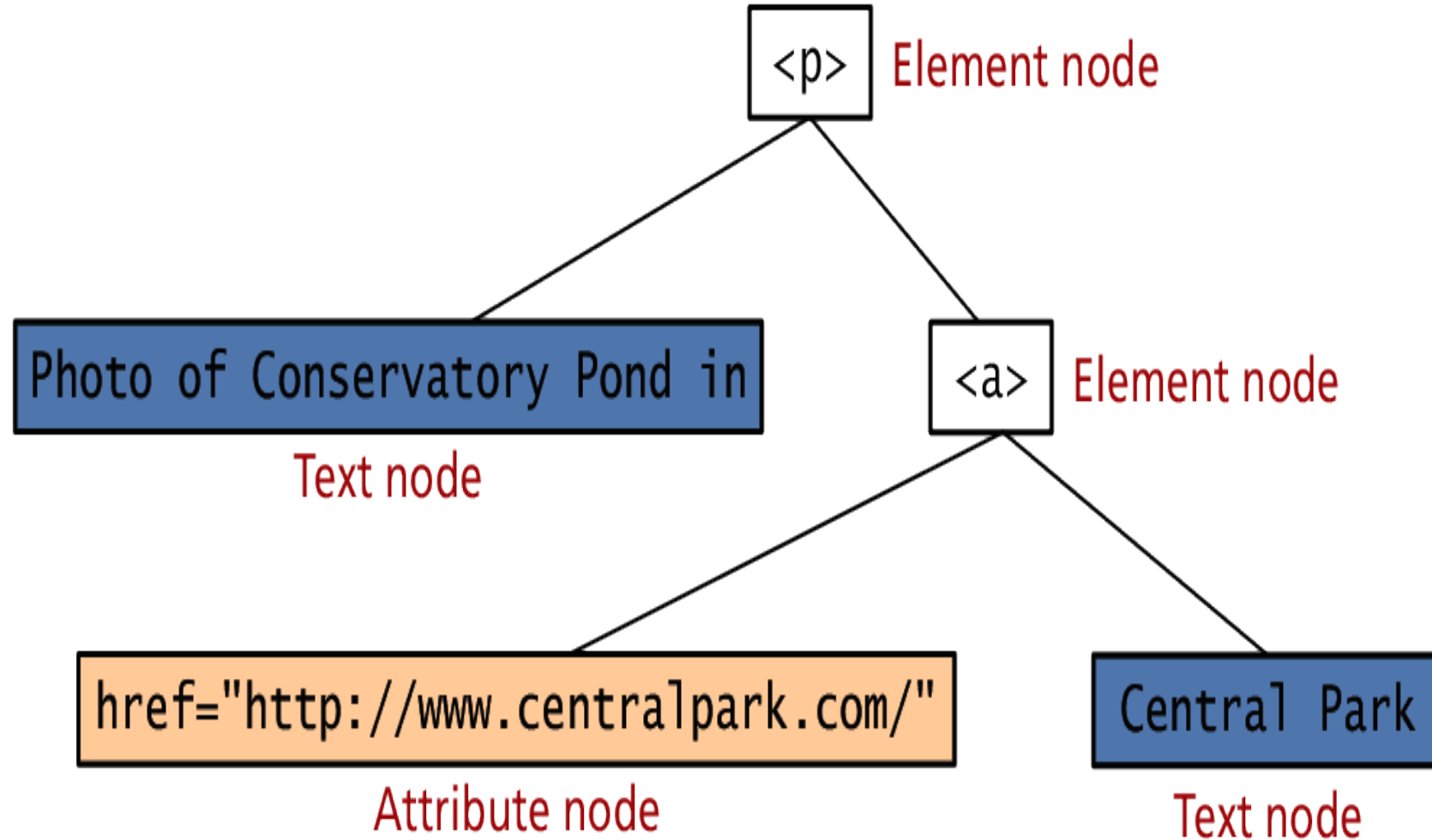
- We already know all about the DOM, but by another name. The HTML tree structure is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.



# 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.

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



Property	Description
attributes	Collection of node attributes
childNodes	A <b>NodeList</b> 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

- The **DOM document object** is the root JavaScript object representing the entire HTML document.
- It contains some properties and methods that are used extensively in 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;`



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 <b>id</b> attribute matches the passed <b>id</b> parameter.
<code>getElementsByName(name)</code>	Returns a <code>nodeList</code> of elements whose tag name matches the passed <b>name</b> parameter.

# Elements

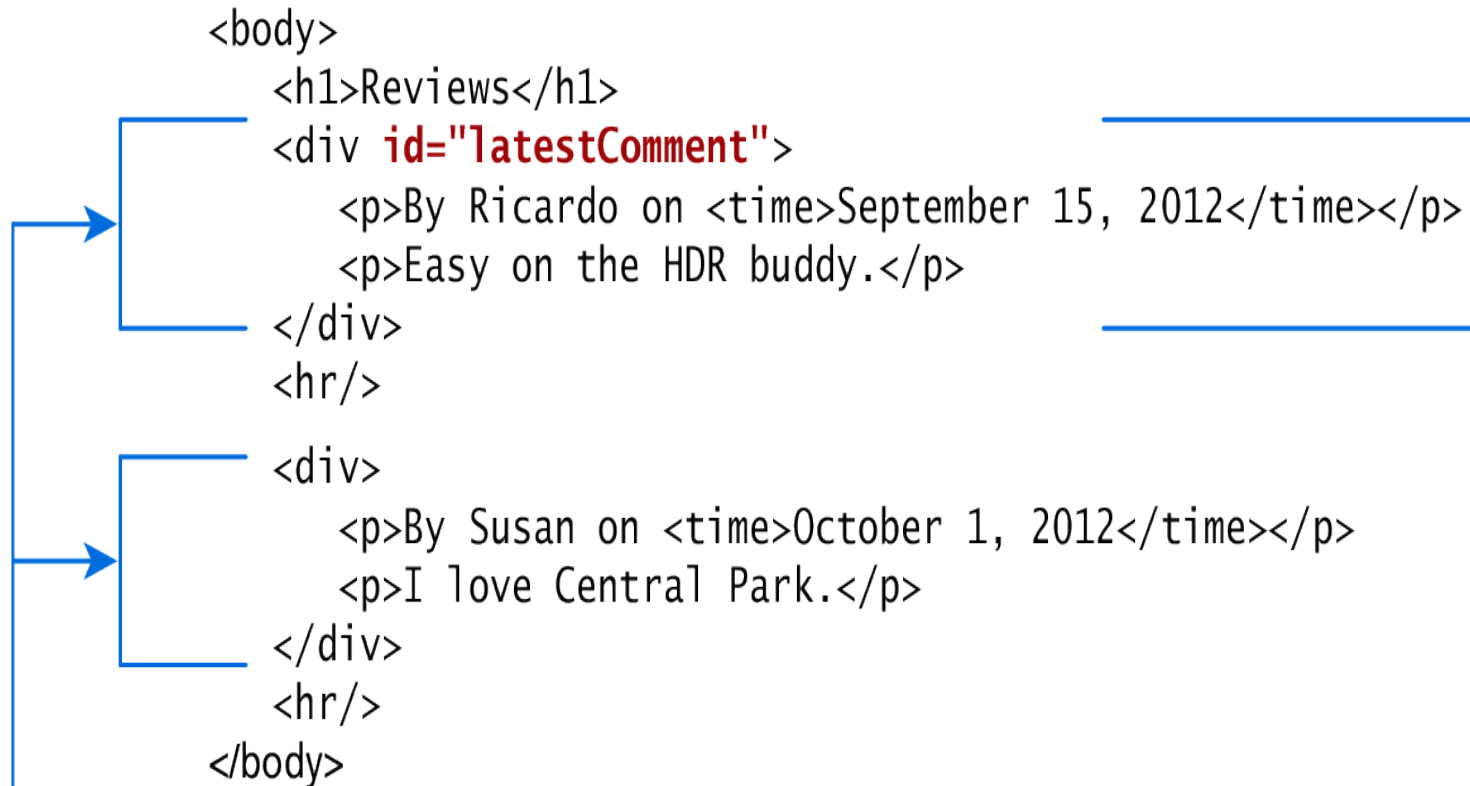
- **var btn = document.createElement("BUTTON");**
- // Create a <button> element  
**var t = document.createTextNode("CLICK ME");**
- // Create a text node  
**btn.appendChild(t);**
- // Append the text to <button>  
**document.body.appendChild(btn);**
- // Append <button> to <body>

# Attribute

- `var h1 = document.getElementsByTagName("H1")[0];`
- `// Get the first <h1> element in the document`  
`var att = document.createAttribute("class");`
- `// Create a "class" attribute`  
`att.value = "democlass";`
- `// Set the value of the class attribute`  
`h1.setAttributeNode(att);`

# Accessing Nodes

```
var abc = document.getElementById("latestComment",
```



```
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

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>";
```

- 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));
```



# Changing 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";`

# Changing 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");`

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 to the 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

# Form validation

- Writing code to pre-validate 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.

```
<body>
```

```
<form name="myForm" action="/action_page.php"  
onsubmit="return validateForm()" method="post">
```

```
  Name: <input type="text" name="fname">
```

```
  <input type="submit" value="Submit">
```

```
</form>
```

```
</body>
```

```
<script>
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == "") {
        alert("Name must be filled out");
        return false;
    }
}
</script>
</head>
```

```
<script>
function myFunction() {
    var x, text;
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>
```

# Events

- Onclick
- Onchange
- Oninput
- Onkeydown
- Onkeyup
- Onkeypress
- Ondrag
- Onmouseover



# Try & 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);  
}
```

# Throw

- 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");  
}
```

- 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.

# References

- <https://www.w3schools.com>