Build COMPETENCY

across your TEAM

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Day 4. JavaScript

Smita B Kumar

Google Cloud Platform

Windows Azure

amazon web services APACHE kafka A distributed streaming platform Apache SOFTWARE FOUNDATION APACHE Spark Java hadoop

JavaScript - Topics Overview

- JS Introduction
- JS Language Basics
- JS Objects
- JS Scope
- JS Events
- JS Arrays
- JS Dates
- JS Comparisons
- JS Type Conversion
- JS Errors
- JS Debugging
- JS Versions
- JS JSON
- JS Forms
- JS Forms
- Forms API
- JS AJAX
- JS RegExp

JS Introduction

- 1. Javascript is a general purpose programming language used to add interactivity and life to an otherwise static webpage.
- 2. Technically, Javascript is one implementation of the specification called as ECMAScript , it was developed by Brendan Eich at Netspace in the year 1995, then called as **LiveScript**.
- 3. For marketing reasons, it was later renamed as Javascript(not associated with Java). Javascript runs on all major platforms and is hardware dependent, Javascript programs are executed by Javascript interpreter built into the browser.
- 4. When a HTML page is loaded by a browser, as soon as the <script> element is encountered, the javascript interpreter takes over and the script is parsed line by line.
- **Note:** Javascript is not the only client side script, Microsoft has its owns client side language called as VB Script.

Javascript Implementations

Javascript implements the specification of ECMAScript specification ECMA-262, but it is much more than that. Javascript consists of three distinct parts

The ECMAScript : The specification specified by ECMA-262, it prepares the base upon which more robust scripting language can be created

The Document Object Model(DOM) : The DOM creates a map of the webpage as a hierarchy of nodes, with each HTML element as a Node containing different kind of Data. Script is used to target the specific node and then manipulate it.

The Browser Object Model(BOM) : The BOM is used to control the browser windows and frames, it enables the developer to pop-up new windows, close, resize and move browser windows.Get detailed information about the browser using the navigator object etc.

Note: These concepts will be explained in great detail with examples in later chapters.

Javascript can be added to any webpages using any of the two methods.

1. Inline Javascript
2. External Javascript

Javascript: The <script> Element

1. Javascript can be inserted on any HTML page using the `<script>` element. Here, the <script> tag denotes the beginning and the </script> tag denotes the end Javascript
2. As soon as the <script> element is encountered in the webpages the javascript interpreter takes over and parses the script from top to bottom
3. The element <script> can be used in two ways, either embed the Javascript directly into the webpage or link to a javascript external file..

Inline Javascript

1. To insert javascript inline within a HTML webpage, simply declare the scripts within the <script> and </script> tags.
2. Care must be taken to see that the string </script> must not appear in the code, or else it results in an error as browser assumes it as the closing tag.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">

function Javascript_demo()
{
alert("Welcome to Synergetics!.Keep calm and Javascript on");
}
</script>

</head>
<body>

<input type="button" onclick="Javascript_demo()" value="Click
here" />
</body>
</html>
```

Javascript External Files

1. Another and more efficient method to insert javascript to webpages is by using the <script> element to include Javascript files from an external location.
2. This is done by specifying the URL of the external Javascript files using the attribute `src` within the <script> element.

Example: Javascript - external Files

```
<!DOCTYPE HTML>
<html>
            <head >
            <script src="javascript_file.js">  </script>
            </head>
 </html>
```

JS Language Basics

- Variables
- Comments
- Operators
- Conditionals
- Functions
- Events
- Objects

Javascript Identifiers (Vairables)

- Javascript identifiers can be the name of a variable, function , property, or function argument. It can comprise of one or more characters satisfying the following format
- * All identifiers must begin with a letter, dollar sign or an underscore()
- * The rest of the characters can be letters, dollar sign, underscore or numbers.
- By convention though not strictly enforced, ECMAScript identifiers use camel-case, meaning that first letter begins with lowercase and every additional letter is offset by Capital letters. Eg: letsGo, lifeIsBeautiful

Javascript Vairables

1. Variables are fundamental building blocks in a programming language, its a data item that represents a memory location on the computer.
2. Variables are similar to containers that can contain hold data items like numbers and string. Variables have a name, valueand type
3. Values assigned to variables can change throughout the course of the program, while the values of literals remain constant.
3. Three types of data can be assigned to the variables.

- numeric
- string
- Boolean

Javascript Vairables

Variable	Explanation	Example
String	A sequence of text known as a string. To signify that the value is a string, you must enclose it in quote marks.	var myVariable = 'Bob';
Number	A number. Numbers don't have quotes around them.	var myVariable = 10;
Boolean	A True/False value. The words true and false are special keywords in JS, and don't need quotes.	var myVariable = true;
Array	A structure that allows you to store multiple values in one single reference.	var myVariable = [1,'Bob','Steve',10]; Refer to each member of the array like this: myVariable[0], myVariable[1], etc.
Object	Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn.	var myVariable = document.querySelector('h1'); All of the above examples too.

Javascript Comments

1. The purpose of a comment is to describe or explain the program.Javascript uses C style comments for both single-line and block comments.
2. A single comment starts with two forward slashes.Any text between a (//) and the end of the line is treated as a comment and hence ignored by Javascript.
3. A block comment starts with a forward slash and asterisk (*/) and closing tag (/*)

```
<script>
 // this is single line comment

 /* this is another comment */


  /* this
    is
    a
    multiple
    line
    comment
  */
</script>
```

Javascript Statements and Semicolons

1. All languages are made up of statements, Javascript is made up of statements comprise of expression which are executed from top to bottom.
2. Multiple statements must be seperated using a semicolon (;), though the parsers can determine where the statement ends but its recommended to always use one.
3. Also multiple statements can be combined into a single code block by using the curly braces, starting with opening braces({) and ending with closing braces (})

```
<script>
 var constant = "3.14"   // No semicolon. VALID, but not
recommended
 var constant = "3.14";  // with Semicolon. Perfect!
 var constant = "3.14" document.write("pi =" + constant);
/* Invalid, as no semicolon to seperate two statements  */
 var constant = "3.14";
 document.write("pi =" + constant);
// Valid as semicolon seperates two statements
 if(a>b)
  alert(a); // valid, but error prone
 if(a>b)
 alert(a); alert(b); // invalid
 if(a>b)
 {
 alert(a);
 alert(b);
 } // valid, and as per rules
</script>
```

Javascript whitespaces

1. In JavaScript, whitespace in between the words are ignored.But name of functions such
as onClick() , toUpperCase() etc cannot contain whitespace even though it comprises of more than one word.
2. Whitespace is preserved within double quotes. All keywords, reserved words, statements etc cannot have any whitespace in between words as the word gets distorted.

```
<script>
 var pi = "3.14";
 var         pi   =      "3.14";   // both statements are equivalent

 toUpperCase() // valid
 to  Upper Case() //invalid
</script>
```

Note: Since extra whitespaces get ignored, users can create line breaks, indents or organise programs etc.

Operators

An operator is a mathematical symbol which produces a result based on two values (or variables). In the following table you can see some of the simplest operators, along with some examples to try out in the JavaScript console.

Operator	Explanation	Symbol(s)	Example

Addition	Used to add two numbers together or glue two strings together.	+	6 + 9;
			"Hello " + "world!";

Subtraction, Multiplication, Division	These do what you'd expect them to do in basic math.	-, *, /	9 - 3;
			8 * 2; // multiply in JS is an asterisk
			9 / 3;

Assignment	You've seen this already: it assigns a value to a variable.	=	var myVariable = 'Bob';

Equality	Does a test to see if two values are equal to one another and returns a true/false(Boolean) result.	===	var myVariable = 3;
			myVariable === 4;

Not, Does-not-equal	Returns the logically opposite value of what it precedes; it turns a true into a false, etc. When it is used alongside the Equality operator, the negation operator tests whether two values are *not* equal.	!, !==	The basic expression is true, but the comparison returns falsebecause we've negated it:
			var myVariable = 3;
			!(myVariable === 3);
			Here we are testing "is myVariableNOT equal to 3". This returnsfalsebecause myVariable IS equal to 3.
			var myVariable = 3;
			myVariable !== 3;

Conditionals

Conditionals are code structures which allow you to test if an expression returns true or not, running alternative code revealed by its result. A very common form of conditionals is the if ... else statement. For example:

```
var iceCream =
'chocolate'; if
(iceCream ===
'chocolate') {
alert('Yay, I love
chocolate ice cream!');
} else { alert('Awwww,
but chocolate is my

favorite...'); }
```

Functions

Functions are a way of packaging functionality that you wish to reuse. When you need the procedure you can call a function, with the function name, instead of rewriting the entire code each time. You have already seen some uses of functions above, for example:

```
1.var myVariable = document.querySelector('h1');
2.alert('hello!');
```

These functions, `document.querySelector` and `alert`, are built into the browser for you to use whenever you desire.

JS Core Object

In Javascript , an object is defined as an ***" unordered collection of properties each of which contains a primitive value, object, or function "***

The objects are described by ***properties*** and their behavior is defined by ***methods***. An object is collection of these properties and methods which can be defined and altered and retrieved by the user.

Javascript objects are dynamic in nature, properties and methods can be added and deleted by the user.Each property or method is identified by the the ***name*** that is mapped to a ***value***.

Since all ***Property*** and ***method*** names are strings, it can be said that objects map strings to values. The ***string-to-value*** maps can be called by various names like ***"hash"*** , ***"hashtable"*** , ***"dictionary"***, ***"associative array"*** .

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

JavaScript - The Number Object

Number Properties
Here is a list of each property and their description.

Sr.No	Property & Description
1	**MAX_VALUE**The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	**MIN_VALUE**The smallest possible value a number in JavaScript can have 5E-324
3	**NaN**Equal to a value that is not a number.
4	**NEGATIVE_INFINITY**A value that is less than MIN_VALUE.
5	**POSITIVE_INFINITY**A value that is greater than MAX_VALUE
6	**prototype**A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document
7	**constructor**Returns the function that created this object's instance. By default this is the Number object.

JavaScript Scope

Scope determines the accessibility (visibility) of variables.

JavaScript Function Scope

In JavaScript there are two types of scope:

- Local scope
- Global scope

JavaScript has function scope: Each function creates a new scope.
Scope determines the accessibility (visibility) of these variables.
Variables defined inside a function are not accessible (visible) from outside the function.

JavaScript Scope

Local JavaScript Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.
Local variables have **local scope**: They can only be accessed within the function.
 Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.
Local variables are created when a function starts, and deleted when the function is completed.

```
<!DOCTYPE html>
<html><body>
<h2>JavaScript Scope</h2>
<p>The variable carName belongs to myFunction().</p>
<p id="demo"></p>
<script>
myFunction();
document.getElementById("demo").innerHTML =
"Outside myFunction() carName is " + typeof carName + ".";
function myFunction() {
   var carName = "Volvo";
}
</script>
</body>
</html>
```

JavaScript Scope

Global JavaScript Variables

A variable declared outside a function, becomes **GLOBAL**.

A global variable has **global scope**: All scripts and functions on a web page can access it.

JavaScript Variables

In JavaScript, objects and functions are also variables.

Scope determines the accessibility of variables, objects, and functions from different parts of the code.

```
<!DOCTYPE html>
<html>
<body>
<p>A GLOBAL variable can be accessed from any script or
function.</p>

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

<script>
var carName = "Volvo";
myFunction();

function myFunction() {
   document.getElementById("demo").innerHTML =
   "I can display " + carName;
}
</script>
</body>
</html>
```

JavaScript Scope

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.
This code example will declare a global variable **carName**, even if the value is assigned inside a function.

Strict Mode

All modern browsers support running JavaScript in "Strict Mode".

Global variables are not created automatically in "Strict Mode".

```
<!DOCTYPE html>
<html><body>
<p>
If you assign a value to a variable that has not been
declared,
it will automatically become a GLOBAL variable:
</p>
<p id="demo"></p>
<script>
myFunction();

// code here can use carName as a global variable
document.getElementById("demo").innerHTML = "I
can display " + carName;
function myFunction() {
    carName = "Volvo";
}
</script>
</body></html>
```

JavaScript HTML DOM Events

Input Events

- onblur - When a user leaves an input field
- onchange - When a user changes the content of an input field
- onchange - When a user selects a dropdown value
- onfocus - When an input field gets focus
- onselect - When input text is selected
- onsubmit - When a user clicks the submit button
- onreset - When a user clicks the reset button
- onkeydown - When a user is pressing/holding down a key
- onkeypress - When a user is pressing/holding down a key
- onkeyup - When the user releases a key
- onkeyup - When the user releases a key
- onkeydown vs onkeyup - Both

JavaScript HTML DOM Events

Mouse Events

- onmouseover/onmouseout - When the mouse passes over an element
- onmousedown/onmouseup - When pressing/releasing a mouse button
- onmousedown - When mouse is clicked: Alert which element
- onmousedown - When mouse is clicked: Alert which button
- onmousemove/onmouseout - When moving the mouse pointer over/out of an image
- onmouseover/onmouseout - When moving the mouse over/out of an image
- onmouseover an image map

JavaScript HTML DOM Events

Click Events

- Acting to the onclick event
- onclick - When button is clicked
- ondblclick - When a text is double-clicked

Load Events

- onload - When the page has been loaded
- onload - When an image has been loaded
- onerror - When an error occurs when loading an image
- onunload - When the browser closes the document
- onresize - When the browser window is resized

JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.
An array is a special variable, which can hold more than one value at a time.

Creating an Array

Example :

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

Using the JavaScript Keyword new

Example:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array

You refer to an array element by referring to the **index number**.

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

<h2>JavaScript Arrays</h2>

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

<script>
var cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Date Objects

JavaScript **Date Object** lets us work with dates:

Wed Jul 04 2018 09:19:48 GMT+0530 (India Standard Time)

Year: 2018 Month: 7 Day: 4 Hours: 9 Minutes 19 Seconds: 48

Creating Date Objects

Date objects are created with the new Date() constructor.

There are 4 ways to create a new date object:

- ***new Date()***
  new Date(year, month, day, hours, minutes, seconds, milliseconds)
  new Date(milliseconds)
  new Date(date string)

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

<h2>JavaScript ISO Dates</h2>

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

<script>
var d = new Date("2015-03-25");
document.getElementById("demo").
innerHTML = d;
</script>

</body>
</html>
```

JavaScript Comparison and Logical Operators

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x = 5**, the table below explains the comparison operators:

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

JavaScript Comparison and Logical Operators

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that **x = 6** and **y = 3**, the table explains the logical operators.

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

```
<!DOCTYPE html>
<html>
<body>
<p>Input your age and click the button:</p>

<input id="age" value="18" />

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

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

<script>
function myFunction() {
    var age, voteable;
    age = document.getElementById("age").value;
    voteable = (age < 18) ? "Too young":"Old enough";
    document.getElementById("demo").innerHTML = voteable +
" to vote ";
```

JavaScript Comparison and Logical Operators

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

```
<!DOCTYPE html>
<html><body>
<h2>JavaScript Comparisons</h2>
<p>Input your age and click the button:</p>
<input id="age" value="18" />
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var age, voteable;
   age = Number(document.getElementById("age").value);
   if (isNaN(age)) {
      voteable = "Input is not a number";
   } else {
      voteable = (age < 18) ? "Too young" : "Old enough";
   }
   document.getElementById("demo").innerHTML = voteable;
}
</script></body></html>
```

JavaScript Type Conversion

Number() converts to a Number, String() converts to a String, Boolean() converts to a Boolean.

In JavaScript there are 5 different data types that can contain values:

- string
- number
- boolean
- object
- function

There are 3 types of objects:

- Object
- Date
- Array

And 2 data types that cannot contain values:

- null
- undefined

JavaScript Type Conversion

The typeof Operator

You can use the **typeof** operator to find the data type of a JavaScript variable.

Example :

```
typeof “Smita"                  // Returns "string"
typeof 3.14                     // Returns "number"
typeof NaN                      // Returns "number"
typeof false                    // Returns "boolean"
typeof [1,2,3,4]                // Returns "object"
typeof {name:’Smita', age:34}  // Returns "object"
typeof new Date()               // Returns "object"
typeof function () {}           // Returns "function"
typeof myCar                    // Returns "undefined" *
typeof null                     // Returns "object"
```

```
<!DOCTYPE html>
<html><body>
<h2>The JavaScript typeof Operator</h2>
<p>The typeof operator returns the type of a variable, object, function or expression.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
   typeof “Smita" + "<br>" +
   typeof 3.14 + "<br>" +
   typeof NaN + "<br>" +
   typeof false + "<br>" +
   typeof [1,2,3,4] + "<br>" +
   typeof {name:’Smita', age:34} + "<br>" +
   typeof new Date() + "<br>" +
   typeof function () {} + "<br>" +
   typeof myCar + "<br>" +
   typeof null;
</script></body></html>
```

JavaScript Type Conversion

JavaScript Type Conversion
JavaScript variables can be converted to a new variable and another data type:
•By the use of a JavaScript function
•**Automatically** by JavaScript itself.

- Values in quotes indicate string values.
- **Red values** indicate values (some) programmers might not expect.

Original Value	Converted to Number	Converted to String	Converted to Boolean
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false
1	1	"1"	true
"0"	0	"0"	**true**
"000"	0	"000"	**true**
"1"	1	"1"	true
NaN	NaN	"NaN"	false
Infinity	Infinity	"Infinity"	true
-Infinity	-Infinity	"-Infinity"	true

JavaScript Type Conversion

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

JavaScript Type Conversion
JavaScript variables can be converted to a new variable and another data type:
•By the use of a JavaScript function
•**Automatically** by JavaScript itself.

- Values in quotes indicate string values.
- **Red values** indicate values (some) programmers might not expect.

Original Value	Converted to Number	Converted to String	Converted to Boolean
""	**0**	""	**false**
"20"	20	"20"	true
"twenty"	NaN	"twenty"	true
[ ]	**0**	""	true
[20]	**20**	"20"	true
[10,20]	NaN	"10,20"	true
["twenty"]	NaN	"twenty"	true
["ten","twenty"]	NaN	"ten,twenty"	true
function(){}	NaN	"function(){}"	true
{ }	NaN	"[object Object]"	true
null	**0**	"null"	false
undefined	NaN	"undefined"	false

JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a **search pattern**. When you search for data in a text, you can use this search pattern to describe what you are searching for.
A regular expression can be a single character, or a more complicated pattern.
Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

/pattern/modifiers;

Example:

var pattern = /mywebsite/i;

mywebsite is a pattern (to be used in a search).
i *is a modifier (modifies the search to be case-insensitive).*

Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

JavaScript Regular Expressions

Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

```
<!DOCTYPE html><html><body>
<h2>JavaScript Regular Expressions</h2>
<p>Search a string for "myWebsite", and display the position of the match:</p>
<p id="demo"></p>
<script>
var str = "Visit myWebsite!";
var n = str.search(/myWebsite/i);
document.getElementById("demo").innerHTML = n;
</script>
<h2>JavaScript String Methods</h2>
<p>Replace "Microsoft" with "myWebsite" in the paragraph below:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo">Please visit Microsoft!</p>
<script>
function myFunction() {
   var str = document.getElementById("demo").innerHTML;
   var txt = str.replace("Microsoft","myWebsite");
   document.getElementById("demo").innerHTML = txt;
}</script>
</body></html>
```

JavaScript Errors - Throw and Try to Catch

JavaScript try and catch

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements **try** and **catch** come in pairs:

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

JavaScript Errors - Throw and Try to Catch

The **try** statement lets you test a block of code for errors.
The **catch** statement lets you handle the error.
The **throw** statement lets you create custom errors.
The **finally** statement lets you execute code, after try and catch, regardless of the result.
Errors Will Happen!
When executing JavaScript code, different errors can occur.
Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

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

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

<script>
try {
   adddlert("Welcome guest!");
}
catch(err) {
   document.getElementById("demo").innerHTML =
err.message;
}
</script>

</body>
</html>
```

JavaScript Errors - Throws Errors

When an error occurs, JavaScript will normally stop and generate an error message.
The technical term for this is:
JavaScript will **throw an exception (throw an error)**.

The throw Statement
The **throw** statement allows you to create a custom error.
Technically you can **throw an exception (throw an error)**.
The exception can be a JavaScript String, a Number, a Boolean or an Object:

```
<!DOCTYPE html><html><body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
<script>
function myFunction() {
   var message, x;
   message = document.getElementById("p01");
   message.innerHTML = "";
   x = document.getElementById("demo").value;
   try {
      if(x == "")  throw "empty";
      if(isNaN(x)) throw "not a number";
      x = Number(x);
      if(x < 5)    throw "too low";
      if(x > 10)   throw "too high";
   }
   catch(err) {
      message.innerHTML = "Input is " + err;
   }
}
</script></body></html>
```

JavaScript Errors - The finally Statement

The **finally** statement lets you execute code, after try and catch, regardless of the result:

```
<!DOCTYPE html><html><body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
<script>
function myFunction() {
   var message, x;
   message = document.getElementById("p01");
   message.innerHTML = "";
   x = document.getElementById("demo").value;
   try {
      if(x == "")  throw "is empty";
      if(isNaN(x)) throw "is not a number";
      x = Number(x);
      if(x > 10)   throw "is too high";
      if(x < 5)    throw "is too low";
   }
   catch(err) {
      message.innerHTML = "Input " + err;
   }
   finally {
      document.getElementById("demo").value = "";
   }
}
</script></body></html>
```

JavaScript Errors - The Error Object

JavaScript has a built in error object that provides error information when an error occurs.
The error object provides two useful properties: name and message.

Error Name Values
Six different values can be returned by the error name property:

Error Object Properties

Property	Description
name	Sets or returns an error name
message	Sets or returns an error message (a string)

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

JavaScript Debugging

“

Errors can (will) happen, every time you write some new computer code.

Code Debugging

Programming code might contain syntax errors, or logical errors.
Many of these errors are difficult to diagnose.
Often, when programming code contains errors, nothing will happen.
There are no error messages, and you will get no indications where to search for errors.
Searching for (and fixing) errors in programming code is called code debugging.

JavaScript Debugging

JavaScript Debuggers

Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.

Built-in debuggers can be turned on and off, forcing errors to be reported to the user.

With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.

Normally, otherwise follow the steps at the bottom of this page, you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

JavaScript Debugging

The console.log() Method

If your browser supports debugging, you can use console.log() to display JavaScript values in the debugger window:

```
<!DOCTYPE html>
<html>
<body>
<h2>My First Web Page</h2>

<p>
Activate debugging in your browser (Chrome, IE, Firefox) with F12, and select
"Console" in the debugger menu.
</p>

<script>
a = 5;
b = 6;
c = a + b;
console.log(c);
</script>
</body>
</html>
```

JavaScript Debugging

Setting Breakpoints

In the debugger window, you can set breakpoints in the JavaScript code.

At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values.

After examining values, you can resume the execution of code (typically with a play button).

The debugger Keyword

The **debugger** keyword stops the execution of JavaScript, and calls (if available) the debugging function.

This has the same function as setting a breakpoint in the debugger.

If no debugging is available, the debugger statement has no effect.

With the debugger turned on, this code will stop executing before it executes the third line.

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

<p>With the debugger turned on, the code below should stop executing before it executes the third line.</p>

<script>
var x = 15 * 5;
debugger;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

JavaScript Versions

JavaScript was invented by ***Brendan Eich*** in 1995, and became an ECMA standard in 1997.
ECMA-262 is the official name of the standard.
ECMAScript is the official name of the language.

- ECMAScript 6 is also called ECMAScript 2015.
- ECMAScript 7 is also called ECMAScript 2016.
- ECMAScript 8 is also called ECMAScript 2017.

ECMAScript Editions

Year	Name	Description
1997	ECMAScript 1	First Edition.
1998	ECMAScript 2	Editorial changes only.
1999	ECMAScript 3	Added Regular Expressions. Added try/catch.
	ECMAScript 4	Was never released.
2009	ECMAScript 5	Added "strict mode". Added JSON support.
2011	ECMAScript 5.1	Editorial changes.
2015	ECMAScript 6	Many new features. Read more in JS Version 6.
2016	ECMAScript 7	Added exponential operator (**). Added Array.prototype.includes.
2017	ECMAScript 8	Added string padding. Added new Object properies. Added Async functions. Added Shared Memory.

Javascript JSON

JSON is a format for storing and transporting data.
JSON is often used when data is sent from a server to a web page.
What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is lightweight data interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand

* The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

Javascript JSON

JSON Example

```
{
"employees":[
    {"firstName":“Smita", "lastName":“Kumar"},
    {"firstName":“Meenal", "lastName":“Sen"},
    {"firstName":“Amit", "lastName":“Gupta"}
]
}
```

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

Javascript JSON

Converting a JSON Text to a JavaScript Object
A common use of JSON is to read data from a web server, and display the data in a web page.
For simplicity, this can be demonstrated using a string as input.
First, create a JavaScript string containing JSON syntax:

```
<!DOCTYPE html>
<html>
<body>
<h2>Create Object from JSON String</h2>
<p id="demo"></p>
<script>
var text = '{"employees":[' +
'{"firstName":“Smita","lastName":“Kumar" },' +
'{"firstName":“Meenal","lastName":“Sen" },' +
'{"firstName":“Amit","lastName":“Gupta" }]}';

obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
</body>
</html>
```

JavaScript Form Validation

HTML form validation can be done by JavaScript. If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

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

JavaScript Form API

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Constraint Validation DOM Methods

Property	Description
checkValidity()	Returns true if an input element contains valid data.
setCustomValidity()	Sets the validationMessage property of an input element.

Constraint Validation DOM Properties

Property	Description
validity	Contains boolean properties related to the validity of an input element.
validationMessage	Contains the message a browser will display when the validity is false.
willValidate	Indicates if an input element will be validated.

JavaScript Form API

```
<!DOCTYPE html>
<html><body>
<p>Enter a number and click OK:</p>
<input id="id1" type="number" min="100" max="300"
required>
<button onclick="myFunction()">OK</button>
<p>If the number is less than 100 or greater than 300, an error
message will be displayed.</p>
<p id="demo"></p>
<script>
function myFunction() {
   var inpObj = document.getElementById("id1");
   if (!inpObj.checkValidity()) {
      document.getElementById("demo").innerHTML =
inpObj.validationMessage;
   } else {
      document.getElementById("demo").innerHTML = "Input
OK";
   }
}
</script></body></html>
```

JavaScript Form API- Validity Properties

Property	Description
customError	Set to true, if a custom validity message is set.
patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
stepMismatch	Set to true, if an element's value is invalid per its step attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.
typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.
valid	Set to true, if an element's value is valid.

JavaScript Form API- Validity Properties

```
<!DOCTYPE html>
<html><body>
<p>Enter a number and click OK:</p>
<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>
<p>If the number is greater than 100 (the input's max
attribute), an error message will be displayed.</p>
<p id="demo"></p>
<script>
function myFunction() {
   var txt = "";
   if (document.getElementById("id1").validity.rangeOverflow) {
      txt = "Value too large";
   } else {
      txt = "Input OK";
   }
   document.getElementById("demo").innerHTML = txt;
}
</script></body></html>
```

AJAX Introduction

- AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.
- AJAX is not a programming language.
- AJAX just uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server) JavaScript and HTML DOM (to display or use the data)
  - AJAX is not a programming language.
  - AJAX is a technique for accessing web servers from a web page.
  - AJAX stands for Asynchronous JavaScript And XML.

AJAX is a developer's dream, because you can:

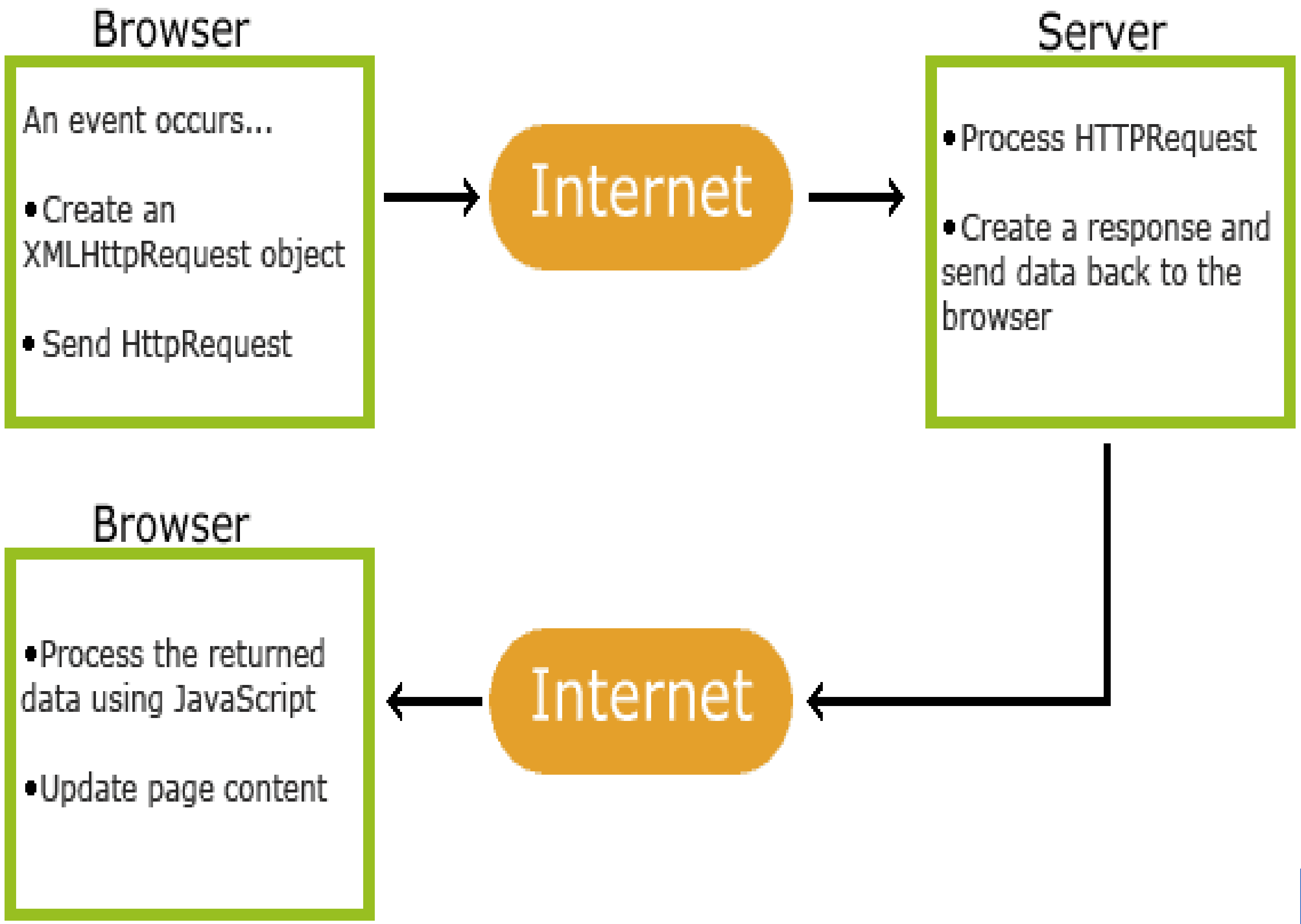
•Read data from a web server - after the page has loaded

•Update a web page without reloading the page

•Send data to a web server - in the background

How AJAX Works

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Internet
Browser
•Process the returned data using JavaScript
•Update page content

AJAX Example

```
<!DOCTYPE html>
<html><body>
<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script></body></html>
```

The HTML page contains a <div> section and a <button>.
The <div> section is used to display information from a server.
The <button> calls a function (if it is clicked).

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Q & A

Contact: smitakumar@synergetics-india.com

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Thank You

Contact: smitakumar@synergetics-india.com